CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases

Tao Yu[†] Rui Zhang[†] He Yang Er[†] Suyi Li[†] Eric Xue[†] Bo Pang[†]
Xi Victoria Lin[¶] Yi Chern Tan[†] Tianze Shi[§] Zihan Li[‡] Youxuan Jiang[‡]
Michihiro Yasunaga[†] Sungrok Shim[†] Tao Chen[†] Alexander Fabbri[†]
Zifan Li[†] Luyao Chen[‡] Yuwen Zhang[‡] Shreya Dixit[†] Vincent Zhang[†]
Caiming Xiong[¶] Richard Socher[¶] Walter S. Lasecki[‡] Dragomir Radev[†]
[†]Yale University [¶]Salesforce Research
[‡]University of Michigan [§]Cornell University
{tao.yu, r.zhang, dragomir.radev}@yale.edu
{xilin, cxiong, rsocher}@salesforce.com

Abstract

We present CoSQL, a corpus for building cross-domain, general-purpose database (DB) querying dialogue systems. It consists of 30k+ turns plus 10k+ annotated SQL queries, obtained from a Wizard-of-Oz (WOZ) collection of 3k dialogues querying 200 complex DBs spanning 138 domains. Each dialogue simulates a real-world DB query scenario with a crowd worker as a user exploring the DB and a SQL expert retrieving answers with SQL, clarifying ambiguous questions, or otherwise informing of unanswerable questions. When user questions are answerable by SQL, the expert describes the SQL and execution results to the user, hence maintaining a natural interaction flow. CoSQL introduces new challenges compared to existing task-oriented dialogue datasets: (1) the dialogue states are grounded in SQL, a domain-independent executable representation, instead of domain-specific slot-value pairs, and (2) because testing is done on unseen databases, success requires generalizing to new domains. CoSQL includes three tasks: SQL-grounded dialogue state tracking, response generation from query results, and user dialogue act prediction. We evaluate a set of strong baselines for each task and show that CoSQL presents significant challenges for future research. The dataset, baselines, and leaderboard will be released at https://yale-lily.github.io/cosql.

1 Introduction

Natural language interfaces to databases (NLIDB) have been studied extensively, with a multitude of different approaches introduced over the past few decades. To this end, considerable progress has been made in querying data via natural language (NL). However, most NL query systems expect the

D_1 : Database about student dormitories containing 5 tables

Q_1 : What are the names of all the dorms? INFORM_SQL

S_1 : `SELECT dorm_name FROM dorm`

A_1 : (Result table with many entries)

R_1 : This is the list of the names of all the dorms. CONFIRM_SQL

Q_2 : Which of those dorms have a TV lounge? INFORM_SQL

S_2 : `SELECT T1.dorm_name FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2.dormid JOIN dorm_amenity AS T3 ON T2.amenid = T3.amenid WHERE T3.amenity_name = 'TV Lounge'`

A_2 : (Result table with many entries)

R_2 : This shows the names of dorms with TV lounges. CONFIRM_SQL

Q_3 : What dorms have no study rooms as amenities? AMBIGUOUS

R_3 : Do you mean among those with TV Lounges? CLARIFY

Q_4 : Yes. AFFIRM

S_4 : `SELECT T1.dorm_name FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2.dormid JOIN dorm_amenity AS T3 ON T2.amenid = T3.amenid WHERE T3.amenity_name = 'TV Lounge' EXCEPT SELECT T1.dorm_name FROM dorm AS T1 JOIN has_amenity AS T2 ON T1.dormid = T2.dormid JOIN dorm_amenity AS T3 ON T2.amenid = T3.amenid WHERE T3.amenity_name = 'Study Room'`

A_4 : Fawlty Towers

R_4 : Fawlty Towers is the name of the dorm that has a TV lounge but not a study room as an amenity. CONFIRM_SQL

. . .

Q_8 : Thanks! THANK_YOU

R_8 : You are welcome. WELCOME

Figure 1: A dialog from the CoSQL dataset. Gray boxes separate the user inputs (Q_i) querying the database (D_i) from the SQL queries (S_i), returned answers (A_i), and expert responses (R_i). Users send an input to the expert, who writes the corresponding SQL query (only seen by the expert) if possible and sends an answer and response description back. Dialogue acts are on the right-hand side (e.g., Q_3 is "ambiguous" and R_3 is "clarify").

query to be well-formed and stated in a single sentence (Zelle and Mooney, 1996; Li and Jagadish, 2014; Yaghmazadeh et al., 2017; Iyer et al., 2017; Zhong et al., 2017; Xu et al., 2017; Shi et al., 2018;

Wang et al., 2018; Yu et al., 2018b,c). In reality, complex questions are usually answered through interactive exchanges (Figure 1). Even for simple queries, people tend to explore the database by asking multiple basic, interrelated questions (Hale, 2006; Levy, 2008; Frank, 2013; Iyyer et al., 2017). This requires systems capable of sequentially processing conversational requests to access information in relational databases. To drive the progress of building a context-dependent NL query system, corpora such as ATIS (Hemphill et al., 1990; Dahl et al., 1994) and SParC (Yu et al., 2019)[1] have been released. However, these corpora assume all user questions can be mapped into SQL queries and do not include system responses.

Furthermore, in many cases, multi-turn interaction between users and NL systems is needed to clarify ambiguous questions (e.g., Q_3 and R_3 in Figure 1), verify returned results, and notify users of unanswerable or unrelated questions. Therefore, a robust dialogue-based NL query agent that can engage with users by forming its own responses has become an increasingly necessary component for the query process. Such systems have already been studied under task-oriented dialogue settings by virtue of continuous effort of corpus creation (Seneff and Polifroni, 2000; Walker et al., 2002; Raux et al., 2005; Mrksic et al., 2015; Asri et al., 2017; Budzianowski et al., 2018) and modelling innovation (Artzi and Zettlemoyer, 2011; Henderson et al., 2013; Lee and Dernoncourt, 2016; hao Su et al., 2016; Dhingra et al., 2016; Li et al., 2016; Mrksic et al., 2017). The goal of these systems is to help users accomplish a specific task, such as flight or hotel booking or transportation planning. However, to achieve these goals, task-oriented dialogue systems rely on pre-defined slots and values for request processing (which can be represented using simple SQL queries consisting of `SELECT` and `WHERE` clauses). Thus, these systems only operate on a small number of domains and have difficulty capturing the diverse semantics of practical user questions.

In contrast, the goal of dialogue-based NLIDB systems is to support general-purpose exploration and querying of databases by end users. To do so, these systems must possess the ability to (1) detect questions answerable by SQL, (2) ground user questions into executable SQL queries if possible, (3) return results to the user in a way that is easily understood and verifiable, and (4) handle unanswerable questions. The difficulty of constructing dialogue-based NLIDB systems stems from these requirements. To enable modeling advances in this field, we introduce CoSQL, the first large-scale cross-domain **Co**nversational text-to-**SQL** corpus collected under the WOZ setting (Budzianowski et al., 2018). CoSQL contains 3,007 dialogues (more than 30k turns with annotated dialogue acts and 10k expert-labeled SQL queries) querying 200 complex DBs spanning across 138 different domains. For each dialogue, we follow the WOZ set-up that involves a crowd worker as a DB user and a college computer science student who is familiar with SQL as an expert (§3).

Like Spider[2] (Yu et al., 2018c) and SParC (Yu et al., 2019), the cross-domain setting in CoSQL enables us to test the ability of systems to generalize on querying different domains via dialogues. We split the dataset in a way that each database only appears in one of train, development, or test set. This setting requires systems to generalize to new domains without additional annotation.

More importantly, unlike most prior work in text-to-SQL systems, CoSQL demonstrates greater language diversity and more frequent user focus changes. It also includes a significant amount of questions that require user clarification and questions that cannot be mapped to SQL queries, introducing the potential to evaluate text-to-SQL dialog act prediction. These features pose new challenges for text-to-SQL systems. Moreover, CoSQL includes system responses that describe SQL queries and the returned results in a way that is easy for users with different backgrounds to understand and verify, as faithful and comprehensible presentation of query results is a crucial component of any NLIDB system.[3]

We introduce three challenge tasks on CoSQL: (1) **SQL-grounded dialogue state tracking** to map user utterances into SQL queries if possible given the interaction history (§5.1), (2) **natural language response generation** based on an executed SQL and its results for user verification

[1]SParC task is available at `https://yale-lily.github.io/sparc`

[2]Spider task is available at `https://yale-lily.github.io/spider`

[3]The DB community has developed query visualization (Li and Jagadish, 2014) and other techniques to provide faithful explanation of a SQL query. These explanations are complementary to the NL ones used in our work and future NLIDB systems could integrate them.

(§5.2) and (3) **user dialogue act prediction** to detect and resolve ambiguous and unanswerable questions (§5.3). We provide detailed data analysis and qualitative examples (§4). For each of the three tasks, we benchmark several competitive baseline models (§6). The performances of these models indicate plenty of room for improvement.

2 Related Work

Text-to-SQL generation Text-to-SQL generation has been studied for decades in both DB and NLP communities. (Warren and Pereira, 1982; Zettlemoyer and Collins, 2005; Popescu et al., 2003; Li et al., 2006; Li and Jagadish, 2014; Iyer et al., 2017; Zhong et al., 2017; Xu et al., 2017; Yu et al., 2018a; Dong and Lapata, 2018; Finegan-Dollak et al., 2018; Guo et al., 2019; Bogin et al., 2019). However, the majority of previous work focus on converting a single, complex question into its corresponding SQL query. Only a few datasets have been constructed for the purpose of mapping context-dependent questions to structured queries. Price (1990); Dahl et al. (1994) collected ATIS that includes series of questions from users interacting with a flight database. Yu et al. (2019) introduced SParC, a large cross-domain semantic parsing in context dataset, consisting of 4k question sequences with 12k questions annotated with SQL queries over 200 complex DBs. Similar to ATIS, SParC includes sequences of questions instead of conversational interactions. An NL question and its corresponding SQL annotation in SParC are constructed by the same expert. Recent works (Suhr et al., 2018; Zhang et al., 2019) have built context-dependent text-to-SQL systems on top of these datasets.

In contrast, CoSQL was collected under a WOZ setting involving interactions between two parties, which contributes to its diverse semantics and discourse covering most types of conversational DB querying interactions (e.g. the system will ask for clarification of ambiguous questions, or inform the user of unanswerable and irrelevant questions). Also, CoSQL includes a natural language system response for the user to understand and verify the systems actions.

Task-oriented dialog systems Task-oriented dialog systems (Henderson et al., 2014; Wen et al., 2016; Mrkšić et al., 2017; Budzianowski et al., 2018) have attracted increasing attention especially due to their commercial values. The goal is to help users accomplish a specific task such as hotel reservation, flight booking, or travel information. These systems (Bordes and Weston, 2017; Zhong et al., 2018; Wu et al., 2019) often predefine slot templates grounded to domain-specific ontology, limiting the ability to generalize to unseen domains. In comparison, our work is to build a system for general-purpose DB exploration and querying. The domain-independent intent representation (SQL query) enables the trained system to work on unseen domains (DB schemas).

While most task-oriented dialog systems need to actively poke the user for information to fill in pre-defined slot-value pairs, the primary goal of system responses in CoSQL is to offer users a reliable way to understand and verify the returned results. If a question can be converted into a SQL query, the user is shown the execution result and the system will describe the SQL query and the result in natural language. In case sthe user questions are ambiguous or unanswerable by SQL, the system either requests the user to rephrase or informs them to ask other questions.

Data-to-Text generation Response generation in CoSQL takes a structured SQL query and its corresponding result table to generate an NL description of the system's interpretation of the user request. Compared to most dialogue-act-to-text generation tasks, the richer semantics of SQL queries makes our task more challenging – besides generating natural and coherent descriptions, faithfully preserving the logic of a SQL query in an NL response is also crucial in our task. Furthermore, this component is related to previous work on text generation from structured data (McKeown, 1985; Iyer et al., 2016; Wiseman et al., 2017).

3 Data Collection

We follow the Wizard-of-Oz setup which facilitates dialogues between DB users and SQL experts to create CoSQL. We recruited Amazon Mechanical Turkers (AMT) to act as DB users and trained 25 graduate- and undergraduate-level computer science students proficient in SQL to act as DB experts. The collection interface (Lasecki et al., 2013) is designed to be easy-to-operate for the experts and intuitive for the users. Detailed explanations of the data collection process is provided below.

Reference goal selection We pre-select a reference goal for each dialogue to ensure the interaction is meaningful and to reduce redundancy within the dataset. Users are asked to explore the given DB content to come up with questions that are likely to naturally arise in real-life scenarios and reflect their query intentions as specified by the reference goals. Following Yu et al. (2019), we selected the complex questions classified as medium, hard, and extra hard in Spider (Yu et al., 2018c) as the reference goals.[4] In total, 3,783 questions were selected on 200 databases. After annotation and reviewing, 3,007 of them were finished and kept in the final dataset.

User setup We developed online chatting interfaces to pair the user with the expert (Figure 6 and 7 in Appendix). When a data collection session starts, the user is first shown multiple tables from a DB to which a reference goal is groundedand is required to read through them. Once they have examined the data stored in the tables, the reference goal question will be revealed on the same screen. The user is encouraged to use the goal question as a guide to ask interrelated questions, but is also allowed to ask other questions exploring the DB. We require the user to ask at least 3 questions.[5] In each turn, if the user question can be answered by a SQL query, they will be shown the result table, and the expert will write an NL response interpreting the executed SQL query based on their understanding of the user's query intent (Figure 8 Appendix). If the user question is ambiguous or cannot be answered with SQL, they will receive clarification questions or notice to rephrase from the expert (detailed in expert setup).

Expert setup Within each session, the expert is shown the same DB content and the reference goal as the user (Figure 8 in Appendix). For each dialogue turn, the expert first checks the user question and labels it using a set of pre-defined user dialog action types (DATs, see Table 4). Then the expert sets the DAT of his response according to the user DAT. Both the user and the expert can have multiple DATs labels in each turn. If the user question is answerable in SQL (labeled as `INFORM_SQL`, e.g. Q_1 in Figure 1), the expert writes down the SQL query[6], executes it, checks the result table, and sends the result table to the user. The expert then describes the SQL query and result table in natural language and sends the response. If the user question is ambiguous, the expert needs to write an appropriate response to clarify the ambiguity (labeled as `AMBIGUOUS`, e.g. Q_3 in Figure 1). Some user questions require the expert to infer the answer based on their world knowledge (labeled as `INFER_SQL`, e.g. Q_3 in Figure 10). If the user question cannot be answered by SQL, the expert will inform them to ask well-formed questions (labeled as `NOT_RELATED`, `CANNOT_UNDERSTAND`, or `CANNOT_ANSWER`). In other cases (labeled as `GREETING`, `THANK_YOU`, etc.), the expert responds with general dialogue expressions (Q_8 in Figure 1).

User quality control Because of the real-time dialogue setting and the expensive annotation procedures on the expert side, conducting quality control on user is crucial for our data collection. We use LegionTools[7] (Lasecki et al., 2014) to post our tasks onto AMT and to recruit and route AMT workers for synchronous real time crowd sourcing tasks. We specify that only workers from the U.S. with 95% approval rates are allowed to accept our task. Before proceeding to the chat room, each AMT worker has to go through a tutorial and pass two short questions[8] to test their knowledge about our task. Only the user who passes the quiz proceeds to the chat room. Throughout the data collection, if a user ignores our instructions in a specific turn, we allow the experts to alert the user through chat and label the corresponding turn as DROP. If a user's actions continue to deviate from instructions, the expert can terminate the dialog before it ends. After each dialogue session terminates, we ask the expert to provide a score from 1 to 5 as an evaluation of the user's performance. Dialogues with a score below 3 are dropped and the user will be blocked from future participation.

[4]Yu et al. (2019) also includes 12.9% of the easy questions in Spider in order to increase dataset diversity. In this work we prioritize the complex questions that trigger more interesting interactions and do not include any easy questions.

[5]The worker is paid $1.20 USD for each dialog. To encourage interaction, we offer $0.50 USD bonus for each dialogue if the user asks more than 4 interesting, interrelated questions.

[6]We use the same SQL annotation protocol as Spider (Yu et al., 2018c) to ensure the same SQL pattern was chosen when multiple equivalent queries were available.

[7]`https://www.cromalab.net/LegionTools/`

[8]One is on how to ask interrelated questions and the other is on how to read multiple tables with reference keys.

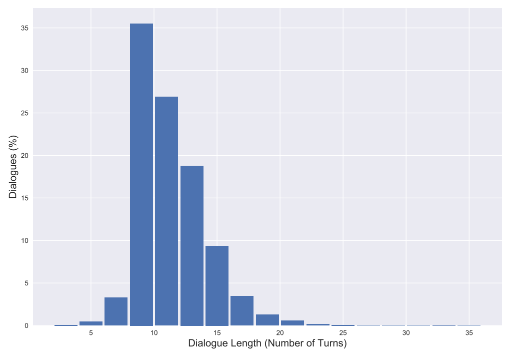


Figure 2: Distributions of dialogue lengths.

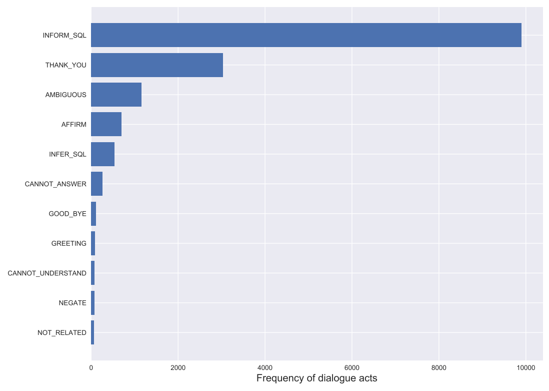


Figure 3: Distributions of user dialog action types.

Data review and post-process We conduct a multi-pass data reviewing process.[9] Two student conducted a first-round review. They focus on correcting any errors in the DATs of the users and the experts, checking if the SQL queries match the user's questions, and modifying or rewriting the expert's responses to contain necessary information in the SQL queries in case they miss any of them. Also, they re-evaluate all dialogues based on the diversity of user questions and reject any dialogues that only contain repeated, simple, and thematically-independent user questions (about 6% of the dialogs). After the first-round review, another two student experts reviewed the refined data to double check the correctness of the DATs, the SQL queries, and the expert responses. They also corrected any grammar errors, and rephrased the user's questions and the expert's responses in a more natural way if necessary. Finally, we ran and parsed all annotated SQL queries to make sure they were executable, following the same annotation protocol as the Spider dataset.

4 Data Statistics and Analysis

We report the statistics of CoSQL and compare it to other task-oriented dialog and context-dependent text-to-SQL datasets. We also conduct detailed analyses on its contextual, cross-domain nature, and question diversity.

Data statistics Table 1 and 2 summarize the statistics of CoSQL. CoSQL contains 3k+ dialogues in total (2,164 in training), which is comparable to or bigger than most commonly used task-oriented dialogue datasets. Figure 2 shows the distribution of dialogue length in the corpus, approximately 80% of dialogues involve 8 or more turns, with a total of 31,148 turns.[10] The average number of tokens in each turn is 11.21. Noticeably, the domain of CoSQL spans over 200 complex databases, overshadowing most other task-oriented dialogue datasets. Comparing to existing context-dependent text-to-SQL datasets, CoSQL contains significantly more turns, out of which 11,039 user utterances are convertible to SQL. In contrast, all NL utterances in ATIS and SParC can be mapped to SQL. CoSQL also has a much larger NL vocabulary.

Dialogue act distribution As shown in Figure 3, CoSQL contains a fairly diverse set of user dialogue action types (DATs). Unsurprisingly, `INFORM_SQL` and `THANK_YOU` are the two most commonly seen DATs. Among the rest of DATs, approximately 40% are `AMBIGUOUS`, demonstrating the paramount importance of system clarification in the DB querying process in general. Another 20% of this subgroup is `INFER_SQL`, which signifies questions that cannot be answered without the aid of human inference.

Semantic complexity As shown in Table 1, if we consider the column names of the 200 DBs of CoSQL as slots and their entries as values, the number of slot-value pairs far exceed those defined in other task-oriented dialogues. Figure 4 shows the total number of occurrences of different SQL keywords in the SQL queries corresponding to these questions. The SQL queries in CoSQL cover all common SQL keywords as well as complicated syntactic structure such as nesting (Fig-

[9]The review interface is shown in Figure 9 (Appendix).

[10]Following Budzianowski et al. (2018), in the statistics report we define the # turns in a dialogue to be the total # messages in the dialogue.

	DSTC2	WOZ 2.0	KVRET	MultiWOZ	**CoSQL**
# dialogs	1,612	600	2,425	8,438	2,164
Total # turns	23,354	4,472	12,732	115,424	22,422
Total # tokens	199,431	50,264	102.077	1,520,970	22.8197
Avg. # turns/dialog	14.49	7.45	5.25	13.68	10.36
Avg. # tokens/turn	8.54	11.24	8.02	13.18	11.34
Total # unique tokens	986	2,142	2,842	24,071	7,502
# databases	1	1	1	7	140
# Slots #	8	4	13	25	3,696
# Values #	212	99	1,363	4,510	>1,000,000

Table 1: Comparison of CoSQL to some commonly used task-oriented dialogue datasets. The numbers are computed for the training part of data in consistency with previous work (Budzianowski et al., 2018).

	CoSQL	SParC	ATIS
# Q sequence	3,007	4298	1658
# user questions	15,598*	12,726	11,653
# databases	200	200	1
# tables	1020	1020	27
Avg. Q len	11.2	8.1	10.2
Vocab	9,585	3794	1582
Avg. # Q turns	5.2	3.0	7.0
Unanswerable Q	✓	✗	✗
User intent	✓	✗	✗
System response	✓	✗	✗

Table 2: Comparison of CoSQL with other context-dependent text-to-SQL datasets. The number are computed over the entire datasets. *For CoSQL we count the total # user utterances.

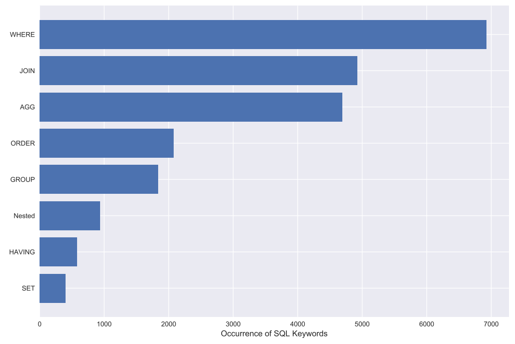


Figure 4: SQL keyword counts.

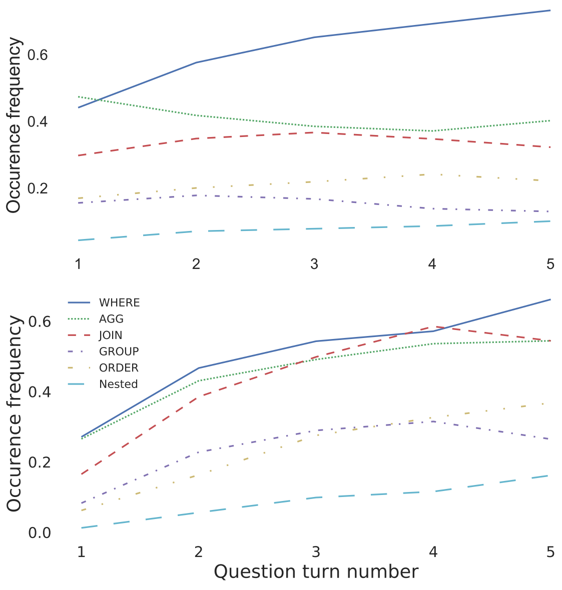


Figure 5: Percentage of question sequences that contain a particular SQL keyword at a specific user utterance turn. The keyword occurrences in CoSQL (upper) slightly fluctuates as the interaction proceeds while that in SParC (lower) demonstrates a clear increasing trend.

ure 5).

Semantic changes by turns We compute the frequency of occurrences of common SQL keywords in different turns for both CoSQL and SParC and compare them in Figure 5 (upper: CoSQL, lower: SParC). Here we count the turn # based on user utterance only. Since CoSQL and SParC span the same domains, Figure 5 reveals a comparison of semantic changes between context-dependent DB questions issued by end users (CoSQL) and expert users (SParC). For CoSQL, the frequencies of all keywords except for `WHERE` do not change significantly throughout the conversation, and the average frequencies of these keywords are in general lower than those of SParC. In addition, `WHERE` occurs slightly more frequently in CoSQL than in SParC. We believe this indicates the exploratory nature of the dialogues we collected, as the users switch their focus more frequently instead of building questions upon previous ones. For example, SQL `AGG` components occur most frequently in the beginning of dialogues, as a result of users familiarizing themselves with the amount of data in the DB or other statistical measures. In contrast, the frequencies of almost all SQL components in SParC increase as the question turn increases. This suggests that questions in SParC have stronger interdependency, as the purpose of this corpus is to study text-to-SQL in context.

Cross domain As shown in Table 3, the dialogues in CoSQL are randomly split into train, development and test sets by DB with a ratio of 7:1:2 (the same split as SParC and Spider).

	Train	Dev	Test
# Dialogs	2164	292	551
# Databases	140	20	40

Table 3: Dataset Split Statistics

5 Tasks and Models

CoSQL is meant to be used as the first benchmark for building general-purpose DB querying dialogue systems in arbitrary domains. Such systems take a user question and determine if it can be answered by SQL (user dialogue act prediction). If the question can be answered by SQL, the system translates it into the corresponding SQL query (SQL-grounded dialogue state tracking), executes the query, returns and shows the result to the user. To improve interpretability and trustworthiness of the result, the system describes the predicted SQL query and result tables to the user for their verification (response generation from SQL and result). Finally, the user checks the results and the system responses and decides if the desired information is obtained or additional questions shall be asked.

Some components relevant to the process above are beyond the scope of our work. First, our response generation task only includes turns where the system's dialogue act is `CONFORM_SQL`. In case the system cannot understand the user's question (the system dialogue act is `CANNOT_ANSWER`) or considers it as unanswerable (`CANNOT_ANSWER`), the system will reply in a standard way to inform the user that it needs clarification or cannot answer that question. The same applies to questions that require human inference (e.g., the system confirms with the user which types of dorms he or she was talking about by asking R_3 instead of immediately translating Q_3 in Figure 1). Currently we do not have a task setup to evaluate the quality of system clarifications. Second, some user questions cannot be directly answered by SQL but are possible to be answered with other type of logical reasoning (e.g., Q_3 in Figure 10). We exclude these questions from our task design and leave them for future research.

5.1 SQL-Grounded dialogue State Tracking

In CoSQL, user dialogue states are grounded in SQL queries. Dialogue state tracking (DST) in this case is to predict the correct SQL query for each user utterance with `INFORM_SQL` label given the interaction context and the DB schema. In our setup, the system does not have access to gold SQL queries from previous turns, which is different from the traditional DST settings in dialogue management where the history of ground-truth dialogue states is given. Comparing to other context-dependent text-to-SQL tasks such as SParC and ATIS, the DST task in CoSQL also include the ambiguous questions if the user affirms the system clarification of them (e.g., Q_4 in Figure 1). In this case, the system clarification is also given as part of the interaction context to predict the SQL query corresponding to the question.[11][12] For instance, to generate S_4 in Figure 1, the input consists of all previous questions (Q_1, Q_2, Q_3), the current user question (Q_4), the DB schema, and the system response R_3.

We benchmark the performance of two strong context-dependent neural text-to-SQL models on this the task, which are the baseline models reported on SParC by Yu et al. (2019).

Context-dependent Seq2Seq (CD-Seq2Seq) The model is originally introduced by (Suhr et al., 2018) for the ATIS task. It incorporates interaction history and is able to copy segments of previous generated SQL queries. Yu et al. (2019) extends it to encode DB schema information such that it works for the cross-domain setting in SParC. We apply the model to our task without any changes.

SyntaxSQL-con SyntaxSQLNet is a SQL-specific syntax-tree based model introduced for Spider (Yu et al., 2018b). Yu et al. (2019) extends it to take previous questions as input when predicting SQL for the current question. We apply the model to our task without any changes.

[11]If a dialogue contains multiple ambiguous questions, the system clarification to all ambiguous questions will be given as input.

[12]The ambiguous questions not confirmed by the user and their system responses are given as part of the conversation history but we do not require a system to predict SQL queries for them.

5.2 Response Generation from SQL and Query Results

This task requires generating a natural language description of the SQL query and the result for each system response labeled as CONFORM_SQL. It considers a SQL query, the execution result, and the DB schema. Preserving logical consistency between SQL and NL response is crucial in this task, in addition to naturalness and syntactical correctness. Unlike other SQL-to-text generation tasks (Xu et al., 2018), our task maps the SQL query to a statement and summarizes the result in that statement (instead of just mapping it back to the user question).

We experiment with three baseline methods for this task.

Template-based Given the SQL and NL response pairs in the training set, we masked variable values in both the SQL and NL response to form parallel SQL-response templates. Given a new SQL query, we employ rule-based approach to select the closest SQL-response template pair from the set. After that, we fill in the selected response template with the columns, tables, and values of the SQL query and the result to generate the final response (see more in Appendix).

Seq2Seq We experiment with a vanilla Seq2Seq model (Sutskever et al., 2014) with attention (Bahdanau et al., 2015), a standard baseline for text generation tasks.

Pointer-generator Oftentimes the column or table names in the NL response are copied from the input SQL query. To capture this phenomenon, we experiment with a pointer-generator network (See et al., 2017), which addresses the problem of out-of-vocabulary word generation in summarization and other text generation tasks. We use a modified version of the implementation from Chen and Bansal (2018).

5.3 User dialogue Act Prediction

For a real-world DB querying dialogue system, it has to decide if the user question can be mapped to a SQL query or if special actions are needed. We define a series of dialogue acts for the DB user and the SQL expert (Table 4).[13] For example, if the user question can be answered by a SQL query, the dialogue act of the question is INFORM_SQL. Since the system DATs are defined in response to the user DATs, our task does not include system dialogue acts prediction.

We experiment with two baseline models for this task.

Majority The dialogue acts of all the user questions are predicted to be the majority dialogue act INFORM_SQL.

TBCNN-pair We employ TBCNN-pair (Mou et al., 2016), a tree-based CNN model with heuristics for predicting entailment and contradiction between sentences. We change the two sentence inputs for the model to a user utterance and the DB schema, and follow the same method in SQL-Net (Xu et al., 2017) to encode each column name.

6 Results and Discussion

SQL-grounded dialog state tracking We use the same evaluation metrics used by the SParC dataset (Yu et al., 2019) to evaluate the model's performance on all questions and interactions (dialogs). The performances of CD-Seq2Seq and SyntaxSQL-con are reported in Table 5. The two models achieve less than 16% question-level accuracy and less than 3% on interaction-level accuracy. Since the two models have been benchmarked on both CoSQL and SParC, we cross-compare their performance on these two datasets. Both models perform significantly worse on CoSQL DST than on SParC. This indicates that CoSQL DST is more difficult than SParC. The possible reasons is that the questions in CoSQL are generated by a more diverse pool of users (crowd workers instead of SQL experts), the task includes ambiguous questions and the context contains more complex intent switches.

Response generation Table 6 shows the results of three different baselines on three metrics: BLEU score (Papineni et al., 2002), logic correctness rate (LCR), and grammar. To compute LCR and grammar score, we randomly sampled 100 descriptions generated by each model. Three students proficient in English participated in the evaluation, They were asked to choose a score 0 or 1 for LCR, and 1 to 5 for grammar check (the larger, the better). For LCR, the final score was decided by majority vote. We computed the average grammar score.

[13] §A.1 defines the complete set of dialogue action types.

Groups	Dialog acts
DB user	`inform_sql`, `infer_sql`, `ambiguous`, `affirm`, `negate`, `not_related`, `cannot_understand`, `cannot_answer`, `greeting`, `goodbye`, `thank_you`
DB expert	`conform_sql`, `clarify`, `reject`, `request_more`, `greeting`, `sorry`, `welcome`, `goodbye`

Table 4: Dialog acts in CoSQL. See § A.1 for the comprehensive definition of each dialogue act.

Model	Question Match		Interaction Match	
	Dev	Test	Dev	Test
CD-Seq2Seq	13.8	13.9	2.1	2.6
SyntaxSQL-con	15.1	14.1	2.7	2.2

Table 5: Performance of various methods over all questions (*question match*) and all interactions (*interaction match*).

Model	BLEU		LCR (%)	Grammar
	Dev	Test	Test	Test
Template	9.5	9.3	41.0	4.0
Seq2Seq	15.3	14.1	27.0	3.5
Pointer-generator	16.4	15.1	35.0	3.6

Table 6: BLEU scores on the development and test sets, and human evaluations of logic correctness rate (LCR) and grammar check on the 100 examples randomly sampled from the test set.

Interestingly, the human evaluation and BLEU scores do not completely agree. While the template-based method is brittle and requires manual effort, it performs significantly better than the two end-to-end neural models in the human evaluation. Because the SQL-question templates provide natural and grammatical sketch of the output, it serves as an advantage in our human evaluation. However, this approach is limited by the small coverage of the training templates and its LCR is only around 40%. On the other hand, the neural models achieve better BLEU scores than the template-based approach. A possible reason for this is that they tend to generate words frequently associated with certain SQL queries. However, the neural models struggle to preserve the SQL query logic in the output. Unsurprisingly, pointer-generator performs better than basic Seq2Seq in terms of both BLEU and human evaluation. The low performances of all methods on LCR show that the task is indeed very challenging.

Model	Dev	Test
Majority	63.3	62.8
TBCNN-pair	84.2	83.9

Table 7: Accuracy of user dialog act prediction on the development and test sets.

User dialog act prediction Table 7 shows the accuracy of the two baselines on predicting user dialog acts. The result of Majority indicates that about 40% of user questions cannot be directly converted into SQL queries. This confirms the necessity of considering a larger set of dialogue actions for building a practical NLIDB system. Even though TBCNN can predict around 85% of user intents correctly, most of the correct predictions are for simple classes such as `INFORM_SQL`, `THANK_YOU`, and `GOODBYE` etc. The F-scores for more interesting and important dialog acts such as `INFER_SQL` and `AMBIGUOUS` are around 10%. This indicates that improving the accuracy on user DAT prediction is still important.

7 Conclusion and Future Work

In this paper, we introduce CoSQL, the first large-scale cross-domain conversational text-to-SQL corpus collected under a Wizard-of-Oz setup. Its language and discourse diversity and cross-domain setting raise exciting open problems for future research. Especially, the baseline model performances on the three challenge tasks suggest plenty space for improvement. The data and challenge leaderboard will be publicly available at `https://yale-lily.github.io/cosql`.

Future Work As discussed in Section 5, some examples in CoSQL include ambiguous and unanswerable user questions and we do not study how a system can effectively clarify those questions or guide the user to ask questions that are answerable. Also, some user questions cannot be answered with SQL but by other forms of logical reasoning the correct answer can be derived. We urge the community to investigate these problems in future work in order to build practical, robust and reliable conversational natural language interfaces to databases.

8 Acknowledgement

This work was supported in part by IBM under the Sapphire Project at the University of Michigan.

References

Yoav Artzi and Luke S. Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *EMNLP*.

Layla El Asri, Hannes Schulz, Shikhar Sharma, Jeremie Zumer, Justin Harris, Emery Fine, Rahul Mehrotra, and Kaheer Suleman. 2017. Frames: A corpus for adding memory to goal-oriented dialogue systems. *CoRR*, abs/1704.00057.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. In *ACL*.

Antoine Bordes and Jason Weston. 2017. Learning end-to-end goal-oriented dialog. *ICLR*.

Pawel Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gasic. 2018. Multiwoz - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *EMNLP*.

Yen-Chun Chen and Mohit Bansal. 2018. Fast abstractive summarization with reinforce-selected sentence rewriting. In *Proceedings of ACL*.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the atis task: The atis-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, Stroudsburg, PA, USA. Association for Computational Linguistics.

Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. 2016. End-to-end reinforcement learning of dialogue agents for information access. In *ACL*.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan Dhanalakshmi Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-sql evaluation methodology. In *ACL 2018*. Association for Computational Linguistics.

Stefan L. Frank. 2013. Uncertainty reduction as a measure of cognitive load in sentence comprehension. *Topics in cognitive science*, 5 3:475–94.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao pei Liu Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *ACL*.

John Hale. 2006. Uncertainty about the rest of the sentence. *Cognitive science*, 30 4:643–72.

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*.

Matthew Henderson, Blaise Thomson, and Jason D. Williams. 2014. The second dialog state tracking challenge. In *SIGDIAL Conference*.

Matthew Henderson, Blaise Thomson, and Steve Young. 2013. Deep neural network approach for the dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 467–471, Metz, France. Association for Computational Linguistics.

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

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany. Association for Computational Linguistics.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831. Association for Computational Linguistics.

Walter Lasecki, Ece Kamar, and Dan Bohus. 2013. Conversations in the crowd: Collecting data for task-oriented dialog learning. In *In Proceedings of the Human Computation Workshop on Scaling Speech and Language Understanding and Dialog through Crowdsourcing at HCOMP 2013*.

Walter S. Lasecki, Mitchell Gordon, Danai Koutra, Malte F. Jung, Steven P. Dow, and Jeffrey P. Bigham. 2014. Glance: Rapidly coding behavioral video with the crowd. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, New York, NY, USA. ACM.

Ji Young Lee and Franck Dernoncourt. 2016. Sequential short-text classification with recurrent and convolutional neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of* *the Association for Computational Linguistics: Human Language Technologies*, pages 515–520, San Diego, California. Association for Computational Linguistics.

Roger P. Levy. 2008. Expectation-based syntactic comprehension. *Cognition*, 106:1126–1177.

Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *VLDB*.

Jiwei Li, Will Monroe, Alan Ritter, Daniel Jurafsky, Michel Galley, and Jianfeng Gao. 2016. Deep reinforcement learning for dialogue generation. In *EMNLP*.

Yunyao Li, Huahai Yang, and HV Jagadish. 2006. Constructing a generic natural language interface for an xml database. In *EDBT*, volume 3896, pages 737–754. Springer.

Kathleen R. McKeown. 1985. *Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, New York, NY, USA.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. 2016. Natural language inference by tree-based convolution and heuristic matching. In *ACL*.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. 2017. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788. Association for Computational Linguistics.

Nikola Mrksic, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Pei hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. 2015. Multi-domain dialog state tracking using recurrent neural networks. In *ACL*.

Nikola Mrksic, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve J. Young. 2017. Neural belief tracker: Data-driven dialogue state tracking.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318. Association for Computational Linguistics.

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.

P. J. Price. 1990. Evaluation of spoken language systems: the atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, pages 91–95.

Antoine Raux, Brian Langner, Dan Bohus, Alan W. Black, and Maxine Eskénazi. 2005. Let's go public! taking a spoken dialog system to the real world. In *INTERSPEECH*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083.

Stephanie Seneff and Joseph Polifroni. 2000. Dialogue management in the mercury flight reservation system. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3*, ANLP/NAACL-ConvSyst '00, pages 11–16.

Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. 2018. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *arXiv preprint arXiv:1809.05054*.

Pei hao Su, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. 2016. Online active reward learning for policy optimisation in spoken dialogue systems. *ACL*.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2238–2249. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Marilyn A. Walker, Alexander I. Rudnicky, Rashmi Prasad, John S. Aberdeen, Elizabeth Owen Bratt, John S. Garofolo, Helen F. Hastie, Audrey N. Le, Bryan L. Pellom, Alexandros Potamianos, Rebecca J. Passonneau, Salim Roukos, Gregory A. Sanders, Stephanie Seneff, and David Stallard. 2002. Darpa communicator: cross-system results for the 2001 evaluation. In *INTERSPEECH*.

Chenglong Wang, Po-Sen Huang, Alex Polozov, Marc Brockschmidt, and Rishabh Singh. 2018. Execution-guided neural program decoding. In *ICML workshop on Neural Abstract Machines and Program Induction v2 (NAMPI)*.

David HD Warren and Fernando CN Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve J. Young. 2016. A network-based end-to-end trainable task-oriented dialogue system. *CoRR*.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263. Association for Computational Linguistics.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. 2019. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Kun Xu, Lingfei Wu, Zhiguo Wang, Mo Yu, Liwei Chen, and Vadim Sheinin. 2018. Sql-to-text generation with graph-to-sequence model. *EMNLP*.

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

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of NAACL*. Association for Computational Linguistics.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018b. Syntaxsqlnet: Syntax tree networks for complex and cross-domain text-to-sql task. In *Proceedings of EMNLP*. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018c. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Irene Li Heyang Er, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Vincent Zhang Jonathan Kraft, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Sparc: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy. Association for Computational Linguistics.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI*, pages 1050–1055, Portland, OR. AAAI Press/MIT Press.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *UAI*.

Rui Zhang, Tao Yu, He Yang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based sql query generation for cross-domain context-dependent questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics.

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

Victor Zhong, Caiming Xiong, and Richard Socher. 2018. Global-locally self-attentive encoder for dialogue state tracking. In *ACL*.

A Appendices

This section provides description of dialog actions in A.1, more details on baseline modifications and hyperparameters in A.2, system response guides in A.3, additional dialog examples in CoSQL dataset in Figure 10 and 11, and the DB user (AMT turkers) and the SQL expert (college computer science students) annotation interfaces in Figure 6, 7, 8, and 9.

A.1 Description of Dialog Acts

For the DB user, we define the following dialog acts:

INFORM_SQL The user informs his/her request if the users question can be answered by SQL. The system needs to write SQL.

INFER_SQL If the users question must be answered by SQL+human inference. For example, users questions are are they..? (yes/no question) or the 3rd oldest.... SQL cannot directly (or unnecessarily complicated) return the answer, but we can infer the answer based on the SQL results.

AMBIGUOUS The users question is ambiguous, the system needs to double check the user's intent (e.g. what/did you mean by...?) or ask for which columns to return.

AFFIRM Affirm something said by the system (user says yes/agree).

NEGATE : Negate something said by the system (user says no/deny).

NOT_RELATED The users question is not related to the database, the system reminds the user.

CANNOT_UNDERSTAND The users question cannot be understood by the system, the system asks the user to rephrase or paraphrase question.

CANNOT_ANSWER The users question cannot be easily answered by SQL, the system tells the user its limitation.

GREETING Greet the system.

GOOD_BYE Say goodbye to the system.

THANK_YOU Thank the system.

For the system, we define the following dialog acts:

CONFIRM_SQL The system creates a natural language response that describes SQL and result table, and asks the user to confirm if the system understood his/her intention.

CLARIFY Ask the user to double check and clarify his/her intention when the users question is ambiguous.

REJECT Tell the user you did not understand/cannot answer his/her question, or the user question is not related.

REQUEST_MORE Ask the user if he/she would like to ask for more info.

GREETING Greet the user.

SORRY Apologize to the user.

WELCOME Tell the user he/she is welcome.

GOOD_BYE Say goodbye to the user.

A.2 Modifications and Hyperparameters for Baselines

CD-Seq2Seq We apply the model with the same settings used in SParC without any changes.

SyntaxSQL-con We apply the model with the same settings used in SParC without any changes.

Template-based We first create a list of SQL query patterns without values, column and table names that cover the most cases in the train set of CoSQL. And then we manually changed the patterns and their corresponding responses to make sure that table, column, and value slots in the responses have one-to-one map to the slots in the SQL query. Once we have the SQL-response mapping list, during the prediction, new SQL statements are compared with every templates to find the best template to use. A score will be computed to represent the similarity between the SQL and each template. The score is computed based on the number of each SQL key components existing in the SQL and each template. Components of the same types are grouped together to allow more flexible matching, like count, max, min are grouped to aggregate. A concrete example of templates is shown: `SELECT column0 FROM table0 WHERE column1 comparison0 value0`. `column0,1` and `table0` represent column name and table name respectively. `comparison0` represents one of the comparison operator including `>=, <=, <,>,=,!=,` and `like`. `value0` represents a value the user uses to constrain the query result.

Seq2Seq We train a word2vec embedding model on the concatenation of the SQL query and response output of the training data for the embedding layer of our Seq2Seq model. We use an embedding dimension of 128, hidden dimension of 256, a single-layer bi-directional LSTM encoder and uni-directional LSTM decoder with attention. We use a batch size of 32, clip the norm of the gradient at 2.0, and do early stopping on the validation loss with a patience of 5. We perform decoding with greedy search.

Pointer-generator We follow the same settings as in the Seq2Seq case with the addition of the copy mechanism during training and testing.

TBCNN-pair The model is modified mainly on the sentence embedding part and classifier part. The input of the modified model is a user utterance and the related column names. Therefore, we replace one of the two sentence embedding modules with a database column name encoding module, which generates representations of the column names related to the sentence. The classifier is modified by adding a label(user dialogue act) number predicting module, which predicts the number of the labels(user dialogue acts) of the user utterance. The label number prediction module is similar to the column number prediction module in SQLNet.

A.3 System Response Guide

System response should be standard and professional. We follow the rules below to write responses for different system dialog action type:

`CLARIFY` ”Did you mean...?”, ”What did you mean by...?”, or anything similar.

`REJECT` ”Sorry, I don’t have the answer to your question...” or anything similar.

`REQUEST_MORE` ”Do you want anything else?” or anything similar.

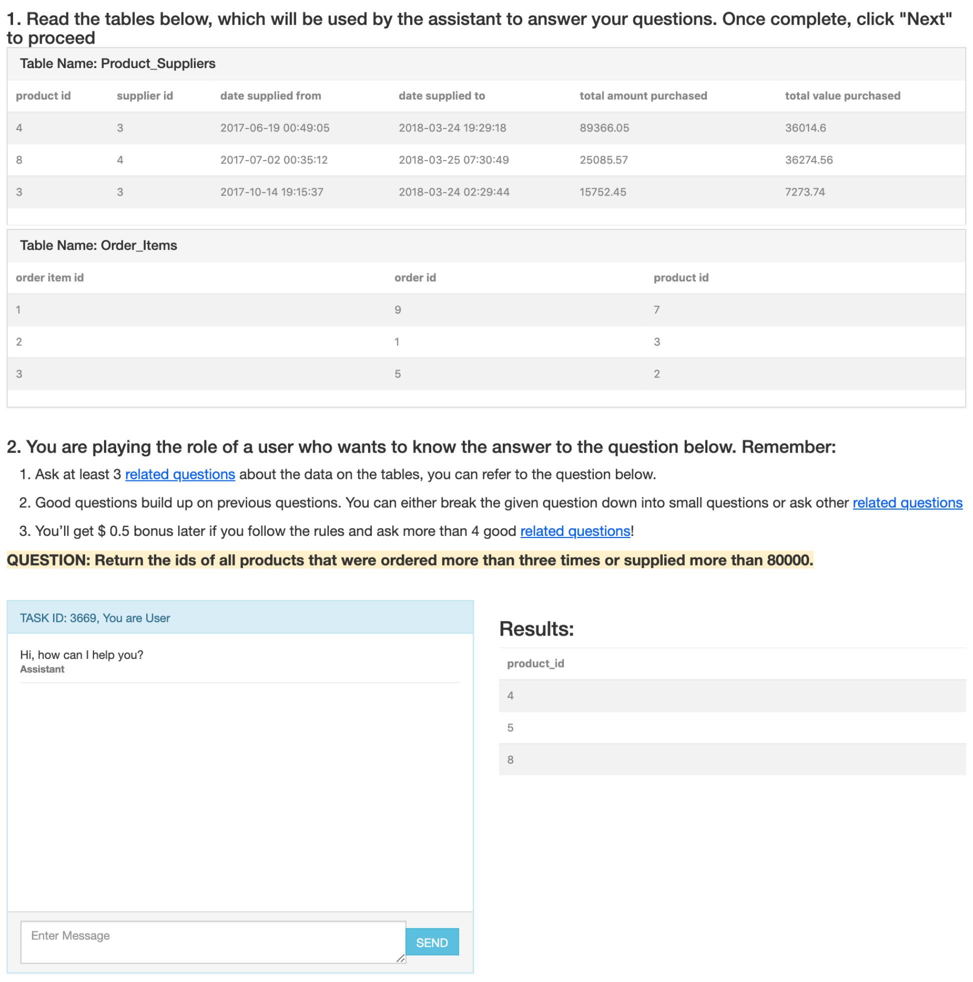
`CONFORM_SQL` We convert SQL written by us back to natural language. (We should use the column names and values in the SQL). Our response has to describe all information in the SQL independently instead of referring to any previous context or subject.

1. If the returned result can be combined with the SQL description, combine them together to generate the response. For example:

   Given SQL:
   SELECT AVG(SALARY) FROM INSTRUCTOR
   Result Returned: 200k
   Your Response:“The average salary of all instructors is 200k.”

2. If the returned result is too large and cannot be combined with the SQL description, describe them separately. For example:

   Given SQL:
   SELECT AVG(T1.SALARY),T1.DEPARTMENT_ID FROM INSTRUCTOR AS T1 JOIN DEPARTMENT AS T2 ON T1.DEPARTMENT_ID = T2.ID GROUP BY T1.DEPARTMENT_ID
   Result Returned: a long table
   Your Response:“Here is the result table that shows the average salary in each department. For example, the average of CS professors is 250k.”



1. Read the tables below, which will be used by the assistant to answer your questions. Once complete, click "Next" to proceed

Table Name: Product_Suppliers

product id	supplier id	date supplied from	date supplied to	total amount purchased	total value purchased
4	3	2017-06-19 00:49:05	2018-03-24 19:29:18	89366.05	36014.6
8	4	2017-07-02 00:35:12	2018-03-25 07:30:49	25085.57	36274.56
3	3	2017-10-14 19:15:37	2018-03-24 02:29:44	15752.45	7273.74

Table Name: Order_Items

order item id	order id	product id
1	9	7
2	1	3
3	5	2

2. You are playing the role of a user who wants to know the answer to the question below. Remember:

1. Ask at least 3 related questions about the data on the tables, you can refer to the question below.
2. Good questions build up on previous questions. You can either break the given question down into small questions or ask other related questions
3. You'll get $ 0.5 bonus later if you follow the rules and ask more than 4 good related questions!

QUESTION: Return the ids of all products that were ordered more than three times or supplied more than 80000.

TASK ID: 3669, You are User

Hi, how can I help you?
Assistant

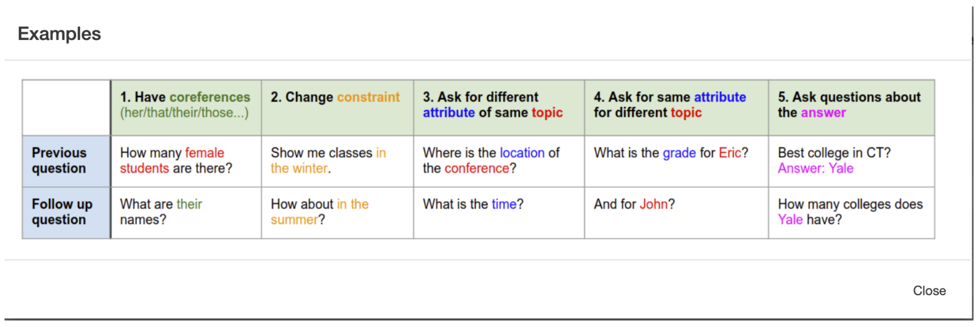
Enter Message

SEND

Results:

product_id
4
5
8

Figure 6: DB User Interface

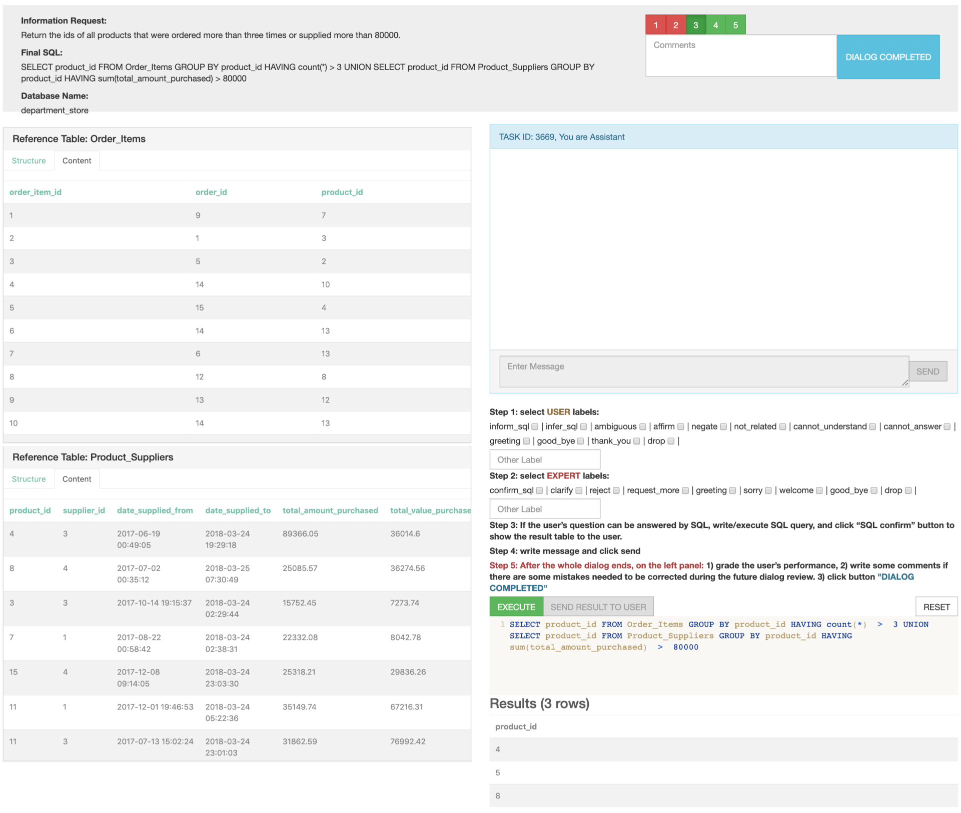


Examples

	1. Have coreferences (her/that/their/those...)	2. Change constraint	3. Ask for different attribute of same topic	4. Ask for same attribute for different topic	5. Ask questions about the answer
Previous question	How many female students are there?	Show me classes in the winter.	Where is the location of the conference?	What is the grade for Eric?	Best college in CT? Answer: Yale
Follow up question	What are their names?	How about in the summer?	What is the time?	And for John?	How many colleges does Yale have?

Close

Figure 7: DB User Related Questions: a pop-up window when the user clicks highlighted "related questions" in the above interface.



Information Request:

Return the ids of all products that were ordered more than three times or supplied more than 80000.

Final SQL:

SELECT product_id FROM Order_Items GROUP BY product_id HAVING count(*) > 3 UNION SELECT product_id FROM Product_Suppliers GROUP BY product_id HAVING sum(total_amount_purchased) > 80000

Database Name:

department_store

Reference Table: Order_Items

order_item_id	order_id	product_id
1	9	7
2	1	3
3	5	2
4	14	10
5	15	4
6	14	13
7	6	13
8	12	8
9	13	12
10	14	13

Reference Table: Product_Suppliers

product_id	supplier_id	date_supplied_from	date_supplied_to	total_amount_purchased	total_value_purchase
4	3	2017-06-19 00:49:05	2018-03-24 19:29:18	89366.05	36014.6
8	4	2017-07-02 00:35:12	2018-03-25 07:30:49	25085.57	36274.56
3	3	2017-10-14 19:15:37	2018-03-24 02:29:44	15752.45	7273.74
7	1	2017-08-22 00:58:42	2018-03-24 02:38:31	22332.08	8042.78
15	4	2017-12-08 09:14:05	2018-03-24 23:03:30	25318.21	29836.26
11	1	2017-12-01 19:46:53	2018-03-24 05:22:36	35149.74	67216.31
11	3	2017-07-13 15:02:24	2018-03-24 23:01:03	31862.59	76992.42

TASK ID: 3669, You are Assistant

Step 1: select USER labels:

inform_sql ☐ | infer_sql ☐ | ambiguous ☐ | affirm ☐ | negate ☐ | not_related ☐ | cannot_understand ☐ | cannot_answer ☐ | greeting ☐ | good_bye ☐ | thank_you ☐ | drop ☐ |

Step 2: select EXPERT labels:

confirm_sql ☐ | clarify ☐ | reject ☐ | request_more ☐ | greeting ☐ | sorry ☐ | welcome ☐ | good_bye ☐ | drop ☐ |

Step 3: If the user's question can be answered by SQL, write/execute SQL query, and click "SQL confirm" button to show the result table to the user.

Step 4: write message and click send

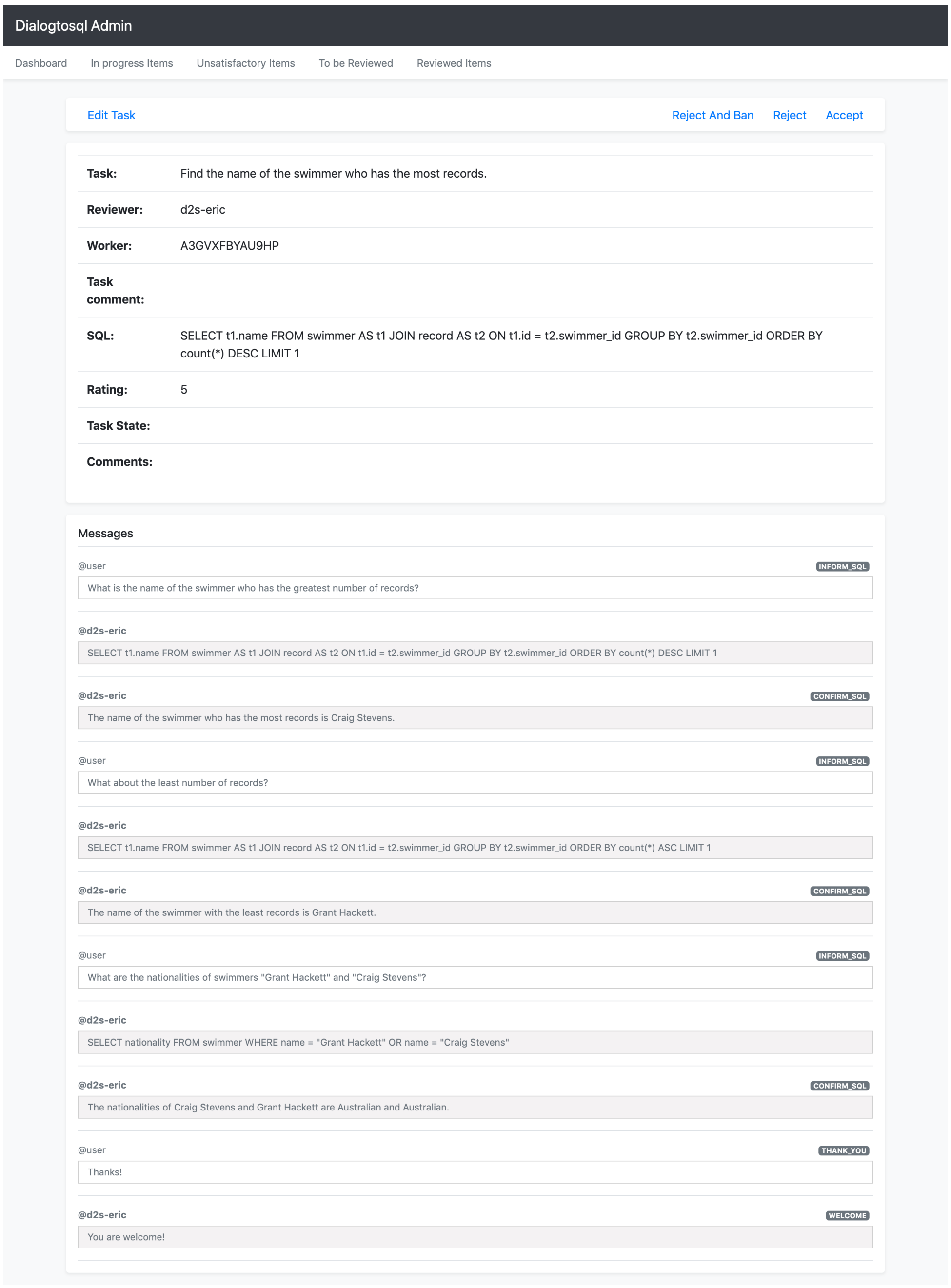
Step 5: After the whole dialog ends, on the left panel: 1) grade the user's performance, 2) write some comments if there are some mistakes needed to be corrected during the future dialog review. 3) click button "DIALOG COMPLETED"

```
SELECT product_id FROM Order_Items GROUP BY product_id HAVING count(*)  >  3 UNION
SELECT product_id FROM Product_Suppliers GROUP BY product_id HAVING
sum(total_amount_purchased)  >  80000
```

Results (3 rows)

product_id
4
5
8

Figure 8: SQL Expert Interface



Task:	Find the name of the swimmer who has the most records.
Reviewer:	d2s-eric
Worker:	A3GVXFBYAU9HP
Task comment:	
SQL:	SELECT t1.name FROM swimmer AS t1 JOIN record AS t2 ON t1.id = t2.swimmer_id GROUP BY t2.swimmer_id ORDER BY count(*) DESC LIMIT 1
Rating:	5
Task State:	
Comments:	

Messages

@user INFORM_SQL
What is the name of the swimmer who has the greatest number of records?

@d2s-eric
SELECT t1.name FROM swimmer AS t1 JOIN record AS t2 ON t1.id = t2.swimmer_id GROUP BY t2.swimmer_id ORDER BY count(*) DESC LIMIT 1

@d2s-eric CONFIRM_SQL
The name of the swimmer who has the most records is Craig Stevens.

@user INFORM_SQL
What about the least number of records?

@d2s-eric
SELECT t1.name FROM swimmer AS t1 JOIN record AS t2 ON t1.id = t2.swimmer_id GROUP BY t2.swimmer_id ORDER BY count(*) ASC LIMIT 1

@d2s-eric CONFIRM_SQL
The name of the swimmer with the least records is Grant Hackett.

@user INFORM_SQL
What are the nationalities of swimmers "Grant Hackett" and "Craig Stevens"?

@d2s-eric
SELECT nationality FROM swimmer WHERE name = "Grant Hackett" OR name = "Craig Stevens"

@d2s-eric CONFIRM_SQL
The nationalities of Craig Stevens and Grant Hackett are Australian and Australian.

@user THANK_YOU
Thanks!

@d2s-eric WELCOME
You are welcome!

Figure 9: Dialogue Review Interface

D_1 : Database about soccer containing 3 tables

. . .

Q_2 : What are their names? INFORM_SQL

S_2 : `SELECT cName FROM College WHERE enr > 18000`

A_2 : OU | FSU

R_2 : The names of the colleges that have enrollment values greater than 18000 are OU and FSU CONFIRM_SQL

Q_3 : Which one has the higher enrollment? INFER_SQL

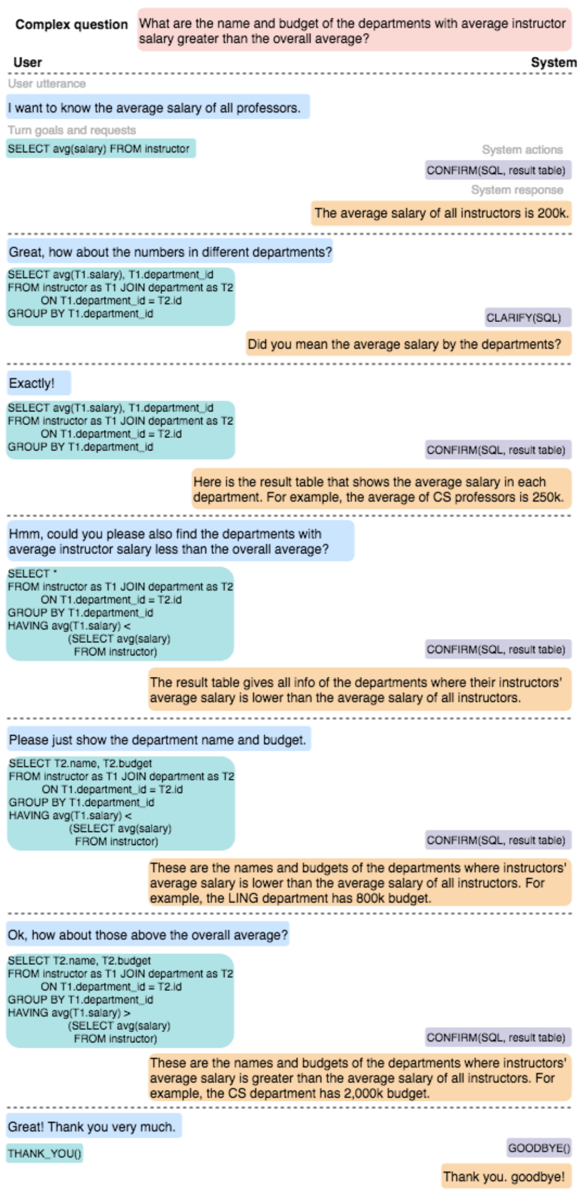
S_3 : `SELECT cName FROM College WHERE enr > 18000 ORDER BY enr DESC LIMIT 1`

A_3 : OU

R_3 : Among the results, OU has a higher enrollment. CONFIRM_SQL

. . .

Figure 10: Part of a dialogue example with INFER_SQL user dialog label



Complex question What are the name and budget of the departments with average instructor salary greater than the overall average?

User **System**

User utterance

I want to know the average salary of all professors.

Turn goals and requests

```
SELECT avg(salary) FROM instructor
```

System actions

CONFIRM(SQL, result table)

System response

The average salary of all instructors is 200k.

Great, how about the numbers in different departments?

```
SELECT avg(T1.salary), T1.department_id
FROM instructor as T1 JOIN department as T2
      ON T1.department_id = T2.id
GROUP BY T1.department_id
```

CLARIFY(SQL)

Did you mean the average salary by the departments?

Exactly!

```
SELECT avg(T1.salary), T1.department_id
FROM instructor as T1 JOIN department as T2
      ON T1.department_id = T2.id
GROUP BY T1.department_id
```

CONFIRM(SQL, result table)

Here is the result table that shows the average salary in each department. For example, the average of CS professors is 250k.

Hmm, could you please also find the departments with average instructor salary less than the overall average?

```
SELECT *
FROM instructor as T1 JOIN department as T2
      ON T1.department_id = T2.id
GROUP BY T1.department_id
HAVING avg(T1.salary) <
             (SELECT avg(salary)
              FROM instructor)
```

CONFIRM(SQL, result table)

The result table gives all info of the departments where their instructors' average salary is lower than the average salary of all instructors.

Please just show the department name and budget.

```
SELECT T2.name, T2.budget
FROM instructor as T1 JOIN department as T2
      ON T1.department_id = T2.id
GROUP BY T1.department_id
HAVING avg(T1.salary) <
             (SELECT avg(salary)
              FROM instructor)
```

CONFIRM(SQL, result table)

These are the names and budgets of the departments where instructors' average salary is lower than the average salary of all instructors. For example, the LING department has 800k budget.

Ok, how about those above the overall average?

```
SELECT T2.name, T2.budget
FROM instructor as T1 JOIN department as T2
      ON T1.department_id = T2.id
GROUP BY T1.department_id
HAVING avg(T1.salary) >
             (SELECT avg(salary)
              FROM instructor)
```

CONFIRM(SQL, result table)

These are the names and budgets of the departments where instructors' average salary is greater than the average salary of all instructors. For example, the CS department has 2,000k budget.

Great! Thank you very much.

THANK_YOU()

GOODBYE()

Thank you. goodbye!

Figure 11: A complete dialogue example