Model-based Interactive Semantic Parsing: A Unified Framework and A Text-to-SQL Case Study

Ziyu Yao[1], Yu Su[1], Huan Sun[1], Wen-tau Yih[2]*
{yao.470, su.809, sun.397}@osu.edu
scottyih@fb.com
[1]The Ohio State University
[2]Facebook AI Research, Seattle

Abstract

As a promising paradigm, *interactive semantic parsing* has shown to improve both semantic parsing accuracy and user confidence in the results. In this paper, we propose a new, unified formulation of the interactive semantic parsing problem, where the goal is to design a *model-based* intelligent agent. The agent maintains its own state as the current predicted semantic parse, decides whether and where human intervention is needed, and generates a clarification question in natural language. A key part of the agent is a world model: it takes a percept (either an initial question or subsequent feedback from the user) and transitions to a new state. We then propose a simple yet remarkably effective instantiation of our framework, demonstrated on two text-to-SQL datasets (WikiSQL and Spider) with different state-of-the-art base semantic parsers. Compared to an existing interactive semantic parsing approach that treats the base parser as a black box, our approach solicits less user feedback but yields higher run-time accuracy.[1]

1 Introduction

Natural language interfaces that allow users to query data and invoke services without programming have been identified as a key application of semantic parsing (Berant et al., 2013; Thomason et al., 2015; Dong and Lapata, 2016; Zhong et al., 2017; Campagna et al., 2017; Su et al., 2017). However, existing semantic parsing technologies often fall short when deployed in practice, facing several challenges: (1) user utterances can be inherently ambiguous or vague, making it difficult to get the correct result in one shot, (2) the accuracy of state-of-the-art semantic parsers are still not high enough for real use, and (3) it is hard for users to validate the semantic parsing results, especially with mainstream neural network models that are known for the lack of interpretability.

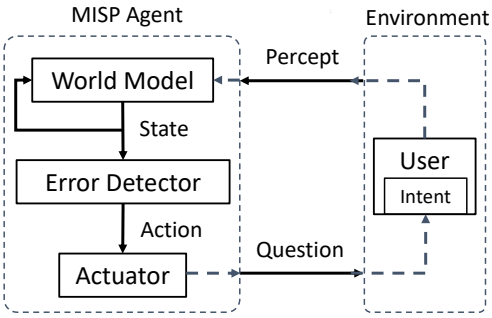

Figure 1: Model-based Interactive Semantic Parsing (MISP) framework.

In response to these challenges, *interactive semantic parsing* has been proposed recently as a practical solution, which includes human users in the loop to resolve utterance ambiguity, boost system accuracy, and improve user confidence via human-machine collaboration (Li and Jagadish, 2014; He et al., 2016; Chaurasia and Mooney, 2017; Su et al., 2018; Gur et al., 2018; Yao et al., 2019). For example, Gur et al. (2018) built the DialSQL system to detect errors in a generated SQL query and request user selection on alternative options via dialogues. Similarly, Chaurasia and Mooney (2017) and Yao et al. (2019) enabled semantic parsers to ask users clarification questions while generating an If-Then program. Su et al. (2018) showed that users overwhelmingly preferred an interactive system over the non-interactive counterpart for natural language interfaces to web APIs. While these recent studies successfully demonstrated the value of interactive semantic parsing in practice, they are often bound to a certain type of formal language or dataset, and the designs are thus ad-hoc and not easily generalizable. For example, DialSQL only applies

*Work started while at AI2

[1]Code available at https://github.com/sunlab-osu/MISP.

to SQL queries on the WikiSQL dataset (Zhong et al., 2017), and it is non-trivial to extend it to other formal languages (e.g., λ-calculus) or even just to more complex SQL queries beyond the templates used to construct the dataset.

Aiming to develop a general principle for building interactive semantic parsing systems, in this work we propose model-based interactive semantic parsing (MISP), where the goal is to design a *model-based intelligent agent* (Russell and Norvig, 2009) that can interact with users to complete a semantic parsing task. Taking an utterance (e.g., a natural language question) as input, the agent forms the semantic parse (e.g., a SQL query) in steps, potentially soliciting user feedback in some steps to correct parsing errors. As illustrated in Figure 1, a MISP agent maintains its *state* as the current semantic parse and, via an *error detector*, decides whether and where human intervention is needed (the *action*). This action is performed by a *question generator* (the *actuator*), which generates and presents to the user a human-understandable question. A core component of the agent is a *world model* (Ha and Schmidhuber, 2018) (hence *model-based*), which incorporates user feedback from the environment and transitions to a new agent state (e.g., an updated semantic parse). This process repeats until a terminal state is reached. Such a design endows a MISP agent with three crucial properties of interactive semantic parsing: (1) being *introspective* of the reasoning process and knowing when it may need human supervision, (2) being able to *solicit user feedback* in a human-friendly way, and (3) being able to *incorporate user feedback* (through state transitions controlled by the world model).

The MISP framework provides several advantages for designing an interactive semantic parser compared to the existing ad-hoc studies. For instance, the whole problem is conceptually reduced to building three key components (i.e., the world model, the error detector, and the actuator), and can be handled and improved separately. While each component may need to be tailored to the specific task, the general framework remains unchanged. In addition, the formulation of a model-based intelligent agent can facilitate the application of other machine learning techniques like reinforcement learning.

To better demonstrate the advantages of the MISP framework, we propose a simple yet remarkably effective instantiation for the text-to-SQL task. We show the effectiveness of the framework based on three base semantic parsers (SQLNet, SQLova and SyntaxSQLNet) and two datasets (WikiSQL and Spider). We empirically verified that with a small amount of targeted, test-time user feedback, interactive semantic parsers improve the accuracy by 10% to 15% absolute. Compared to an existing interactive semantic parsing system, DialSQL (Gur et al., 2018), our approach, despite its much simpler yet more general system design, achieves better parsing accuracy by asking only half as many questions.

2 Background & Related Work

Semantic Parsing. Mapping natural language utterances to their formal semantic representations, semantic parsing has a wide range of applications, including question answering (Berant et al., 2013; Dong and Lapata, 2016; Finegan-Dollak et al., 2018), robot navigation (Artzi and Zettlemoyer, 2013; Thomason et al., 2015) and Web API calling (Quirk et al., 2015; Su et al., 2018). The target application in this work is text-to-SQL, which has been popularized by the WikiSQL dataset (Zhong et al., 2017). One of the top-performing models on WikiSQL is SQLNet (Xu et al., 2017), which leverages the pre-defined SQL grammar sketches on WikiSQL and solves the SQL generation problem via "slot filling." By augmenting SQLNet with a table-aware BERT encoder (Devlin et al., 2019) and by revising the value prediction in `WHERE` clauses, SQLova (Hwang et al., 2019) advances further the state of the art. Contrast to WikiSQL, the recently released Spider dataset (Yu et al., 2018c) focuses on complex SQL queries containing multiple keywords (e.g., `GROUP BY`) and may join multiple tables. To handle such complexity, Yu et al. (2018b) proposed SyntaxSQLNet, a syntax tree network with modular decoders, which generates a SQL query by recursively calling a module following the SQL syntax. However, because of the more realistic and challenging setting in Spider, it only achieves 20% in accuracy.

We experiment our MISP framework with the aforementioned three semantic parsers on both WikiSQL and Spider. The design of MISP allows naturally integrating them as the base parser. For example, when SQLNet fills a sequence of slots to produce a SQL query, a "state" in MISP corresponds to a partially generated SQL query and it

transitions as SQLNet fills the next slot.

Interactive Semantic Parsing. To enhance parsing accuracy and user confidence in practical applications, interactive semantic parsing has emerged as a promising solution (Li and Jagadish, 2014; He et al., 2016; Chaurasia and Mooney, 2017; Su et al., 2018; Gur et al., 2018; Yao et al., 2019). Despite their effectiveness, existing solutions are somewhat ad-hoc and bound to a specific formal language and dataset. For example, DialSQL (Gur et al., 2018) is curated for WikiSQL, where SQL queries all follow the same and given grammar sketch. Similarly, (Yao et al., 2019) relies on a pre-defined two-level hierarchy among components in an If-Then program and cannot generalize to formal languages with a deeper structure. In contrast, MISP aims for a general design principle by explicitly identifying and decoupling important components, such as error detector, question generator and world model. It also attempts to integrate and leverage a strong base semantic parser, and transforms it to a natural interactive semantic parsing system, which substantially reduces the engineering cost.

3 Model-based Interactive Semantic Parsing

We now discuss the MISP framework (Figure 1) in more detail. Specifically, we highlight the function of each major building block and the relationships among them, and leave the description of a concrete embodiment to Section 4.

Environment. The *environment* consists of a user with a certain intent, which corresponds to a semantic parse that the user expects the agent to produce. Based on this intent, the user gives an initial natural language utterance u_0 to start a semantic parsing *session* and responds to any clarification question from the agent with feedback u_t at interaction turn t.

Agent State. The *agent state* s is an agent's internal interpretation of the environment based on all the available information. A straightforward design of the agent state is as the currently predicted semantic parse. It can also be endowed with meta information of the parsing process such as prediction probability or uncertainty to facilitate error detection.

World Model. A key component of a MISP agent is its *world model* (Ha and Schmidhuber, 2018), which compresses the historical percepts throughout the interaction and predicts the future based on the agent's knowledge of the world. More specifically, it models the transition of the agent state, $p(s_{t+1}|s_t, u_t)$, where u_t is the user feedback at step t and s_{t+1} is the new state. The transition can be deterministic or stochastic.

Error Detector. A MISP agent introspects its state and decides whether and where human intervention is needed. The *error detector* serves this role. Given the current state s_t (optionally the entire interaction history) and a set of *terminal states*, it decides on an *action* a_t: If the agent is at a terminal state, it terminates the session, executes the semantic parse, and returns the execution results to the user; otherwise, it determines a span in the current semantic parse that is likely erroneous and passes it, along with necessary context information needed to make sense of the error span, to the actuator.

Actuator. An *actuator* has a user-facing interface and realizes an agent's actions in a user-friendly way. In practice, it can be a natural language generator (NLG) (He et al., 2016; Gur et al., 2018; Yao et al., 2019) or an intuitive graphical user interface (Su et al., 2018; Berant et al., 2019), or the two combined.

4 MISP-SQL: An Instantiation of MISP for Text-to-SQL

Under the MISP framework, we design an interactive semantic parsing system (Figure 2), named MISP-SQL, for the task of text-to-SQL translation. MISP-SQL assumes a base text-to-SQL parser and leverages it to design the world model and the error detector. The world model is essentially a wrapper that takes the user input and changes the behavior of the base semantic parser (e.g., by changing the probability distribution or removing certain prediction paths). The error detector makes decisions based on the *uncertainty* of the predictions: if the parser is uncertain about a prediction, it is more likely to be an error. The actuator is a template-based natural language question generator developed for the general SQL language. Figure 2 shows an example of the MISP-SQL agent.

4.1 Agent State

For ease of discussion, we assume the base parser generates the SQL query by predicting a sequence

of SQL components,[2] as in many state-of-the-art systems (Xu et al., 2017; Wang et al., 2018; Yu et al., 2018a; Hwang et al., 2019). Agent state s_t is thus defined as a *partial* SQL query, i.e., s_t=$\{o_1, o_2, ..., o_t\}$, where o_t is the predicted SQL component at time step t, such as `SELECT place` in Figure 2. What constitutes a SQL component is often defined differently in different semantic parsers, but typically dictated by the SQL syntax. To support introspection and error detection, each prediction is associated with its uncertainty, which is discussed next.

4.2 Error Detector

The error detector in MISP-SQL is introspective and greedy. It is introspective because it examines the uncertainty of the predictions as opposed to the predictions themselves. On the other hand, it is greedy because its decisions are solely based on the last prediction o_t instead of the entire state s_t.

We experiment with two uncertainty measures, based on the probability of o_t estimated by the base semantic parser, as well as its standard deviation under Bayesian dropout (Gal and Ghahramani, 2016), respectively.

Probability-based Uncertainty. Intuitively if the base semantic parser gives a low probability to the top prediction at a step, it is likely uncertain about the prediction. Specifically, we say a prediction o_t needs user clarification if its probability is lower than a threshold p^*, i.e.,

$$p(o_t) < p^*.$$

This strategy is shown to be strong in detecting misclassified and out-of-distribution examples (Hendrycks and Gimpel, 2017).

Dropout-based Uncertainty. Dropout (Srivastava et al., 2014) has been used as a Bayesian approximation for estimating model uncertainty (Gal and Ghahramani, 2016) in several tasks (Dong et al., 2018; Siddhant and Lipton, 2018; Xiao and Wang, 2019). Different from its standard application to prevent models from overfitting in training time, we use it at test time to measure model uncertainty, similar to (Dong et al., 2018). The intuition is that if the probability on a prediction varies dramatically (as measured by the standard deviation) across different perturbations under dropout, the model is likely uncertain about it. Specifically, the uncertainty on prediction o_t is calculated as:

$$\text{STDDEV}\{p(o_t|W_i)\}_{i=1}^{N},$$

where W_i is the parameters of the base semantic parser under the i-th dropout perturbation, and the uncertainty score is the standard deviation of the prediction probabilities over N random passes. We say o_t needs user clarification if its uncertainty score is greater than a threshold s^*.

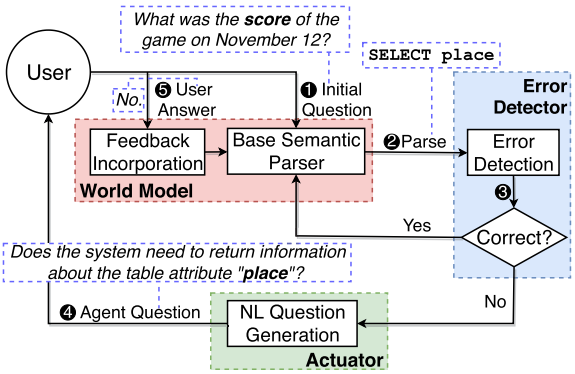

Figure 2: MISP-SQL Agent. The base semantic parser incrementally parses the user question (Step 1) into a SQL query by first selecting a column from the table (Step 2). This partial parse is examined by the error detector (Step 3), who determines that the prediction is incorrect (because the uncertainty is high) and triggers the actuator to ask the user a clarification question (Step 4). The user feedback is then incorporated into the world model (Step 5) to update the agent state. If the prediction was correct, Step 2 would be repeated to continue the parsing.

Terminal State. The only terminal state is when the base semantic parser indicates end of parsing.

4.3 Actuator: An NL Generator

The MISP-SQL agent performs its action (e.g., validating the column "*place*") via asking users *binary* questions, hence the actuator is a natural language generator (NLG). Although there has been work on describing a SQL query with an NL statement (Koutrika et al., 2010; Ngonga Ngomo et al., 2013; Iyer et al., 2016; Xu et al., 2018), few work studies generating *questions* about a certain SQL component in a systematic way.

Inspired by (Koutrika et al., 2010; Wang et al., 2015), we define a rule-based NLG, which consists of a seed lexicon and a grammar for deriving questions. Table 1 shows rules covering

[2] In practice this assumption may not be necessary as long as there is a reasonable way to chunk the semantic parse to calculate uncertainty and formulate clarification questions.

[Lexicon]
is greater than|equals to|is less than → OP[>|=|<]
sum of values in|average value in|number of|minimum value in|maximum value in → AGG[sum|avg|count|min|max]
[Grammar]
"*col*" → COL[*col*]
Does the system need to return information about COL[*col*] ? → Q[*col*‖SELECT *agg*? *col*]
Does the system need to return AGG[*agg*] COL[*col*] ? → Q[*agg*‖SELECT *agg col*]
Does the system need to return a value after any mathematical calculations on COL[*col*] ? → Q[*agg*=None‖SELECT *col*]
Does the system need to consider any conditions about COL[*col*] ? → Q[*col*‖WHERE *col op val*]
The system considers the following condition: COL[*col*] OP[*op*] a value. Is this condition correct? → Q[*op*‖WHERE *col op val*]
The system considers the following condition: COL[*col*] OP[*op*] *val*. Is this condition correct? → Q[*val*‖WHERE *col op val*]

Table 1: Domain-general lexicon and grammar for NL generation in MISP-SQL (illustrated for WikiSQL; a more comprehensive grammar for Spider can be found in Appendix A).

SQL queries on WikiSQL (Zhong et al., 2017). The seed lexicon defines NL descriptions for basic SQL elements in the form of "$n \rightarrow t[p]$", where n is an NL phrase, t is a pre-defined syntactic category and p is either an aggregator (e.g., `avg`) or an operator (e.g., $>$). For example, "is greater than $\rightarrow$ OP[$>$]" specifies a phrase "is greater than" to describe the operator "$>$". In MISP-SQL, we consider four syntactic categories: AGG for aggregators, OP for operators, COL for columns and Q for generated questions. However, it can be extended with more lexicon entries and grammar rules to accommodate more complex SQL in Spider (Yu et al., 2018c), which we show in Appendix A.

The grammar defines rules to derive questions. Each column is described by itself (i.e., the column name). Rules associated with each Q-typed item "Q[v‖`Clause`]" constructs an NL question asking about v in `Clause`. The `Clause` is the necessary context to formulate meaningful questions. Figure 3 shows a derivation example. Note that, both the lexicon and the grammar in our system are domain-agnostic in the sense that it is not specific to any database. Therefore, it can be reused for new domains in the future. Database-specific rules, such as naming each column with a more canonical phrase (rather than the column name), are also possible.

4.4 World Model

The agent incorporates user feedback and updates its state with a world model. Different from DialSQL which trains an additional neural network, the MISP-SQL agent directly employs the base semantic parser to transition states, which saves additional training efforts.

As introduced in Section 4.3, the agent raises a binary question to the user about a predicted SQL component o_t. Therefore, the received user feedback either confirms the prediction or negates it. In the former case, the state is updated by proceeding to the next decoding step, i.e., s_{t+1}={o_1, ..., o_t, o_{t+1}}, where o_{t+1} is the predicted next component and s_{t+1} shows the updated partial parse. In the latter case, the user feedback is incorporated to constrain the search space of the base parser (i.e., forbidding the parser from making the same wrong prediction), based on which the parser refreshes its prediction and forms a new state s_{t+1}={o_1, ..., o_{t-1}, o_{t+1}}, where o_{t+1} is a predicted alternative to replace o_t. To avoid being trapped in a large search space, for each SQL component, we consider a maximum number of K alternatives (in addition to the original prediction) to solicit user feedback on.

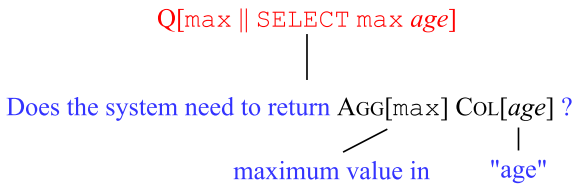

Figure 3: Deriving an NL question about the aggregator `max` in the clause "`SELECT max`(age)" from the rooted Q-typed item .

5 Experiments

We apply our approach to the task of mapping natural language questions to SQL queries. In this section, we first describe the basic setup, including the datasets and the base semantic parsers, followed by the system results on both simulated and real users.

5.1 Experimental Setup

We evaluate our proposed MISP-SQL agent on WikiSQL (Zhong et al., 2017), which contains 80,654 hand-annotated pairs of ⟨NL question,

System	SQLNet Acc_{qm}	SQLNet Acc_{ex}	SQLNet Avg. #q	SQLova Acc_{qm}	SQLova Acc_{ex}	SQLova Avg. #q
no interaction	0.615	0.681	N/A	0.797	0.853	N/A
DialSQL	0.690	N/A	2.4	N/A	N/A	N/A
MISP-SQL$^{\text{Unlimit10}}$	0.932	0.948	7.445	0.985	0.991	6.591
MISP-SQL$^{\text{Unlimit3}}$	0.870	0.900	7.052	0.955	0.974	6.515
MISP-SQL$^{p^*=0.95}$	0.782	0.824	1.713	0.912	0.939	0.773
MISP-SQL$^{p^*=0.8}$	0.729	0.779	1.104	0.880	0.914	0.488

Table 2: Simulation evaluation of MISP-SQL (based on SQLNet or SQLova) on WikiSQL Test set. "MISP-SQL$^{p^*=\text{X}}$" denotes our agent with probability-based error detection (threshold at X). "MISP-SQL$^{\text{UnlimitK}}$" denotes a variant that asks questions for every component, with up to $K+1$ questions per component.

SQL query⟩, distributed across 24,241 tables from Wikipedia. Our experiments follow the same data split as in (Zhong et al., 2017).

We experiment MISP-SQL with two base semantic parsers: SQLNet (Xu et al., 2017) and SQLova (Hwang et al., 2019). Unlike in DialSQL's evaluation (Gur et al., 2018), we do not choose Seq2SQL (Zhong et al., 2017) as a base parser but SQLova instead, because it achieves similar performance as SQLNet while SQLova is currently the best open-sourced model on WikiSQL, which can give us a more comprehensive evaluation. For each of the two base semantic parsers, we test our agent with two kinds of error detectors, based on prediction probability and Bayesian dropout, respectively (Section 4.2). We tune the threshold p^* within $0.5 \sim 0.95$ and s^* within $0.01 \sim 0.2$. Particularly for uncertainty-based detection measured by Bayesian dropout, the number of passes N is set to 10, with a dropout rate 0.1. The dropout layers are applied at the same positions as when each semantic parser is trained. When the agent interacts with users, the maximum number of alternative options (in addition to the original prediction) per component, K, is set to 3. If the user negates all the $K+1$ predicted candidates, the agent will keep the original prediction, as in (Gur et al., 2018).

5.2 Simulation Evaluation

In simulation evaluation, each agent interacts with a *simulated* user, who gives a yes/no answer based on the ground-truth SQL query. If the agent fails to correct its predictions in three consecutive interaction turns, the user will leave the interaction early and the agent has to finish the remaining generation without further help from the user.

Overall Comparison. We first compare MISP-SQL with the two base semantic parsers without interactions in Table 2. For SQLNet, we also compare our system with the reported performance of DialSQL (Gur et al., 2018, Table 4). However, since DialSQL is not open-sourced and it is not easy to reproduce it, we are unable to adapt it to SQLova for more comparisons. Following (Xu et al., 2017; Hwang et al., 2019), we evaluate the SQL query match accuracy ("**Acc_{qm}**", after converting the query into its canonical form) and the execution accuracy ("**Acc_{ex}**") of each agent. "**Avg. #q**" denotes the average number of questions per query. For any base parser, MISP-SQL improves their performance by interacting with users. Particularly for SQLNet, MISP-SQL outperforms the DialSQL system with only *half* the number of questions (1.104 vs. 2.4), and has a much simpler design without the need of training an extra model (besides training the base parser, which DialSQL needs to do as well). Our agent can even boost the strong performance of SQLova from 85% to 94% in execution accuracy, with merely 0.773 questions per query.

We also present an "upper-bounded" accuracy of our agent, when it does not adopt any error detector and asks questions about *every* component with at most 10 ("MISP-SQL$^{\text{Unlimit10}}$") or 3 ("MISP-SQL$^{\text{Unlimit3}}$") alternatives. Interestingly, even for the weaker SQLNet parser, most true predictions have already been contained within the top 10 options (giving 0.932 query match accuracy). When equipped with the stronger SQLova parser, the agent has a potential to boost the execution accuracy to around 100% by considering only the top 3 options of every prediction. The complete results can be found in Appendix B.

Error Detector Comparison. We then compare the probability-based and dropout-based error detectors in Figure 4, where each marker indicates

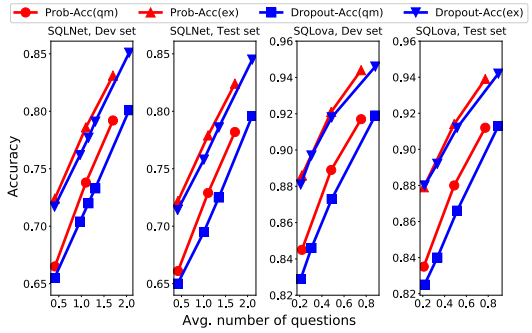

Figure 4: Comparison of probability- and dropout-based error detection.

the agent's accuracy and the average number of questions it needs under a certain error detection threshold. Consistently for both SQLNet and SQLova, the probability-based error detector can achieve the same accuracy with a lower number of questions than the dropout-based detector. It is also observed that this difference is greater in terms of query match accuracy, around 0.15 $\sim$ 0.25 for SQLNet and 0.1 $\sim$ 0.15 for SQLova. A more direct comparison of various settings under the same average number of questions can be found in Appendix C.

To better understand how each kind of error detectors works, we investigate the portion of questions that each detector spends on *right* predictions (denoted as "$\mathbf{Q_r}$"). An ideal system should ask fewer questions on right predictions while identify more truly incorrect predictions to fix the mistakes. We present the question distributions of the various systems in Table 3. One important conclusion drawn from this table is that probability-based error detection is much more effective on identifying incorrect predictions. Consider the system using probability threshold 0.5 for error detection (i.e., "p^*=0.5") and the one using dropout-based error detector with a threshold 0.2 (i.e., "s^*=0.2") on SQLNet. When both systems ask around the same number of questions during the interaction, the former spends only 16.9% of unnecessary questions on correct predictions (Q_r), while the latter asks twice amount of them (32.1%). Similar situations are also observed for SQLova. It is also notable that, when the probability threshold is lower (which results in a fewer total number of questions), the portion of questions on right actions drops significantly (e.g., from 23.0% to 16.9% when the threshold changes from 0.8 to 0.5 on SQLNet). However, this portion remains almost unchanged for dropout-based error detection.

SQLNet			SQLova		
System	**Avg. #q**	**Q_r%**	**System**	**Avg. #q**	**Q_r%**
p^*=0.8	1.099	23.0%	p^*=0.8	0.484	28.9%
p^*=0.5	0.412	16.9%	p^*=0.5	0.220	18.4%
s^*=0.07	1.156	34.5%	s^*=0.03	0.489	50.4%
s^*=0.2	0.406	32.1%	s^*=0.05	0.306	52.5%

Table 3: Portion of interaction questions on right predictions (Q_r%) for each agent setting on WikiSQL Dev set (smaller is better). "p^*/s^*=X" denotes our agent with probability/dropout-based error detection (threshold at X).

5.3 Extend to Complex SQL Generation

A remarkable characteristic of MISP-SQL is its generalizability, as it makes the best use of the base semantic parser and requires no extra model training. To verify it, we further experiment MISP-SQL on the more complex text-to-SQL dataset "Spider" (Yu et al., 2018c). The dataset consists of 10,181 questions on multi-domain databases, where SQL queries can contain complex keywords such as `GROUP BY` or join several tables. We extend the NLG lexicon and grammar (Section 4.3) to accommodate this complexity, with details shown in Appendix A.

We adopt SyntaxSQLNet (Yu et al., 2018b) as the base parser.[3] In our experiments, we follow the same database split as in (Yu et al., 2018c) and report the Exact Matching accuracy ("$\mathbf{Acc_{em}}$") on Dev set.[4] Other experimental setups remain the same as when evaluating MISP-SQL on WikiSQL. Table 4 shows the results.

We first observe that, via interactions with simulated users, MISP-SQL improves SyntaxSQLNet by 10% accuracy with reasonably 3 questions per query. However, we also realize that, unlike on WikiSQL, in this setting, the probability-based error detector requires more questions than the Bayesian uncertainty-based detector. This can be explained by the inferior performance of the base SyntaxSQLNet parser (merely 20% accuracy without interaction). In fact, the portion of questions that the probability-based detector spends on right predictions (Q_r) is still half of that the dropout-based detector asks (12.8% vs. 24.8%). However, it wastes around 60% of questions on unsolvable wrong predictions. This typically hap-

[3] We chose SyntaxSQLNet because it was the best model by the paper submission time. In principle, our framework can also be applied to more sophisticated parsers such as (Bogin et al., 2019; Guo et al., 2019).

[4] We do not report results on Spider test set since it is not publicly available.

System	Acc_{em}	Avg. #q
no interaction	0.190	N/A
MISP-SQL$^{\text{Unlimit10}}$	0.522	14.878
MISP-SQL$^{\text{Unlimit3}}$	0.382	11.055
MISP-SQL$^{p^*=0.95}$	0.300	3.908
MISP-SQL$^{p^*=0.8}$	0.268	3.056
MISP-SQL$^{s^*=0.01}$	0.315	3.815
MISP-SQL$^{s^*=0.03}$	0.290	2.905

Table 4: Simulation evaluation of MISP-SQL (built on SyntaxSQLNet) on Spider Dev set.

pens when the base parser is not strong enough, i.e., cannot rank the true option close to the top, or when there are unsolved wrong precedent predictions (e.g., in "`WHERE` *col op val*", when *col* is wrong, whatever *op*/*val* following it is wrong). This issue can be alleviated when more advanced base parsers are adopted in the future.

5.4 Human Evaluation

We further conduct human user study to evaluate the MISP-SQL agent. Our evaluation setting largely follows Gur et al. (2018). For each base semantic parser, we randomly sample 100 examples from the corresponding dataset (either WikiSQL Test set or Spider Dev set) and ask three human evaluators, who are graduate students with only rudimentary knowledge of SQL based on our survey, to work on each example and then report the averaged results. We present to the evaluators the initial natural language question and allow them to view the table headers to better understand the question intent. On Spider, we also show the name of the database tables. We select error detectors based on the simulation results: For SQLNet and SQLova, we equip the agent with a probability-based error detector (threshold at 0.95); for SyntaxSQLNet, we choose a Bayesian uncertainty-based error detector (threshold at 0.03). As in the simulation evaluation, we cannot directly compare with DialSQL in human evaluation because the code is not yet publicly available.

Table 5 shows the results. In all settings, MISP-SQL improves the base parser's performance, demonstrating the benefit of involving human interaction. However, we also notice that the gain is not as large as in simulation, especially on SQLova. Through interviews with the human evaluators, we found that the major reason is that they sometimes had difficulties understanding the true intent of some test questions that are ambiguous, vague, or contain entities they are not familiar with. We believe this reflects a general challenge of setting up human evaluation for semantic parsing that is close to the real application setting, and thus set forth the following discussion.

System	$Acc_{qm/em}$	Acc_{ex}	Avg. #q
SQLNet			
no interaction	0.580	0.660	N/A
MISP-SQL (simulation)	0.770	0.810	1.800
MISP-SQL (real user)	0.633	0.717	1.510
SQLova			
no interaction	0.830	0.890	N/A
MISP-SQL (simulation)	0.920	0.950	0.550
MISP-SQL (real user)	0.837	0.880	0.533
+ w/ full info.	0.907	0.937	0.547
SyntaxSQLNet			
no interaction	0.180	N/A	N/A
MISP-SQL (simulation)	0.290	N/A	2.730
MISP-SQL (real user)	0.230	N/A	2.647

Table 5: Human evaluation on 100 random examples for MISP-SQL agents based on SQLNet, SQLova and SyntaxSQLNet, respectively.

5.5 Discussion on Future Human Evaluation

Most human evaluation studies for (interactive) semantic parsers so far (Chaurasia and Mooney, 2017; Gur et al., 2018; Su et al., 2018; Yao et al., 2019) use pre-existing test questions (e.g., from datasets like WikiSQL). However, this introduces an undesired discrepancy, that is, human evaluators may not necessarily be able to understand the true intent of the given questions in an faithful way, especially when the question is ambiguous, vague, or containing unfamiliar entities.

This discrepancy is clearly manifested in our human evaluation with SQLova (Table 5). When the base parser is strong, many of the remaining incorrectly parsed questions are challenging not only for the base parser but also for human evaluators. We manually examined the situations where evaluators made a different choice than the simulator and found that 80% of such choices happened when the initial question is ambiguous or the gold SQL annotation is wrong. For example, for the question "*name the city for kanjiža*" it is unlikely for human evaluators to know that "kanjiža" is an "Urban Settlement" without looking at the table content or knowing the specific background knowledge beforehand. This issue has also been reported as the main limitation to further improve SQLova (Hwang et al., 2019), which

could in principle be resolved by human interactions if the users have a clear and consistent intent in mind. To verify this, we conduct an additional experiment with SQLova where human evaluators can view the table content as well as the gold SQL query before starting the interaction to better understand the true intent (denoted as "w/ full info" in Table 5). As expected, the MISP-SQL agent performs much better (close to simulation) when users know what they are asking. It further confirms that a non-negligible part of the accuracy gap between simulation and human evaluation is due to human evaluators not fully understanding the question intent and giving false feedback.

To alleviate this discrepancy, a common practice is to show human evaluators the schema of the underlying database, as Gur et al. (2018) and we did (Section 5.4), but it is still insufficient, especially for entity-related issues (e.g., "kanjiža"). On the other hand, while exposing human evaluators to table content helps resolve the entity-related issues, it is likely to introduce undesired biases in favor of the system under test (i.e., "overexposure"), since human evaluators may then be able to give more informative feedback than real users.

To further reduce the discrepancy between human evaluation and real use cases, one possible solution is to ask human evaluators to come up with questions from scratch (instead of using pre-existing test questions), which guarantees intent understanding. While this solution may still require exposure of table content to evaluators (such that they can have some sense of each table attribute), overexposure can be mitigated by showing them only part (e.g., just a few rows) of the table content, similar to the annotation strategy by Zhong et al. (2017). Furthermore, the reduced controllability on the complexity of the evaluator-composed questions can be compensated by conducting human evaluation in a larger scale. We plan to explore this setting in future work.

6 Conclusion and Future Work

This work proposes a new and unified framework for the interactive semantic parsing task, named MISP, and instantiates it successfully on the text-to-SQL task. We outline several future directions to further improve MISP-SQL and develop MISP systems for other semantic parsing tasks:

Improving Agent Components. The flexibility of MISP allows improving on each agent component separately. Take the error detector for example. One can augment the probability-based error detector in MISP-SQL with probability calibration, which has been shown useful in aligning model confidence with its reliability (Guo et al., 2017). One can also use learning-based approaches, such as a reinforced decision policy (Yao et al., 2019), to increase the rate of identifying wrong and solvable predictions.

Lifelong Learning for Semantic Parsing. Learning from user feedback is a promising solution for lifelong semantic parser improvement (Iyer et al., 2017; Padmakumar et al., 2017; Labutov et al., 2018). However, this may lead to a non-stationary environment (e.g., changing state transition) from the perspective of the agent, making its training (e.g., error detector learning) unstable. In the context of dialog systems, Padmakumar et al. (2017) suggests that this effect can be mitigated by jointly updating the dialog policy and the semantic parser batchwisely. We leave exploring this aspect in our task to future work.

Scaling Up. It is important for MISP agents to scale to larger backend data sources (e.g., knowledge bases like Freebase or Wikidata). To this end, one can improve MISP from at least three aspects: (1) using more intelligent interaction designs (e.g., free-form text as user feedback) to speed up the hypothesis space searching globally, (2) strengthening the world model to nail down a smaller set of plausible hypotheses based on both the initial question and user feedback, and (3) training the agent to learn to improve the parsing accuracy while minimizing the number of required human interventions over time.

Acknowledgments

This research was sponsored in part by the Army Research Office under cooperative agreements W911NF-17-1-0412, NSF Grant IIS1815674, Fujitsu gift grant, and Ohio Supercomputer Center (Center, 1987). 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 Office 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

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Jonathan Berant, Daniel Deutch, Amir Globerson, Tova Milo, and Tomer Wolfson. 2019. Explaining queries over web tables to non-experts. In *Proceedings of the 35th IEEE International Conference on Data Engineering (ICDE)*.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-SQL parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565.

Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S Lam. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *Proceedings of the 26th International Conference on World Wide Web*, pages 341–350. International World Wide Web Conferences Steering Committee.

Ohio Supercomputer Center. 1987. Ohio supercomputer center. http://osc.edu/ark:/19495/f5s1ph73.

Shobhit Chaurasia and Raymond J Mooney. 2017. Dialog for language to code. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 175–180.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 33–43.

Li Dong, Chris Quirk, and Mirella Lapata. 2018. Confidence modeling for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 743–753.

Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535.

Izzeddin Gur, Semih Yavuz, Yu Su, and Xifeng Yan. 2018. DialSQL: Dialogue based structured query generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1339–1349.

David Ha and Jürgen Schmidhuber. 2018. World models. *ArXiv preprint arXiv:1803.10122*.

Luheng He, Julian Michael, Mike Lewis, and Luke Zettlemoyer. 2016. Human-in-the-loop parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2337–2342.

Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *Proceedings of International Conference on Learning Representations*.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on WikiSQL with table-aware word contextualization. *ArXiv preprint arXiv:1902.01069*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973.

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)*, volume 1, pages 2073–2083.

Georgia Koutrika, Alkis Simitsis, and Yannis E Ioannidis. 2010. Explaining structured queries in natural language. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 333–344. IEEE.

Igor Labutov, Bishan Yang, and Tom Mitchell. 2018. Learning to learn semantic parsers from natural language supervision. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1676–1690.

Fei Li and HV Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. 2013. Sorry, I don't speak SPARQL: translating SPARQL queries into natural language. In *Proceedings of the 22nd international conference on World Wide Web*, pages 977–988. ACM.

Aishwarya Padmakumar, Jesse Thomason, and Raymond J Mooney. 2017. Integrated learning of dialog strategies and semantic parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 547–557.

Chris Quirk, Raymond Mooney, and Michel Galley. 2015. Language to code: Learning semantic parsers for If-This-Then-That recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 878–888.

Stuart Russell and Peter Norvig. 2009. Artificial intelligence: A modern approach.

Aditya Siddhant and Zachary C Lipton. 2018. Deep Bayesian active learning for natural language processing: Results of a large-scale empirical study. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2904–2909.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, Michael Gamon, and Mark Encarnacion. 2017. Building natural language interfaces to Web APIs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 177–186. ACM.

Yu Su, Ahmed Hassan Awadallah, Miaosen Wang, and Ryen W White. 2018. Natural language interfaces with fine-grained user interaction: A case study on web APIs. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Jesse Thomason, Shiqi Zhang, Raymond J Mooney, and Peter Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. 2018. Pointing out SQL queries from text.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1332–1342.

Yijun Xiao and William Yang Wang. 2019. Quantifying uncertainties in natural language processing tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7322–7329.

Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. 2018. SQL-to-text generation with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 931–936.

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

Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, and Huan Sun. 2019. Interactive semantic parsing for if-then recipes via hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2547–2554.

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 the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594.

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 the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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*, pages 3911–3921.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *ArXiv preprint arXiv:1709.00103*.

A Extension to Complex SQL

Table 8 shows the extended lexicon entries and grammar rules in NLG for applying our MISP-SQL agent to generate more complex SQL queries, such as those on Spider (Yu et al., 2018c). In this dataset, a SQL query can associate with multiple tables. Therefore, we name a column by combining the column name with its table name (i.e., "*col*" in table "*tab*" $\rightarrow$ COL[*col* (table *tab*)]). For simplicity, we omit "(table *tab*)" when referring to a column *col* in the grammar.

B Simulation Evaluation Results

The complete simulation experiment results of MISP-SQL agents (based on SQLNet and SQLova) are shown in Table 6 & 7.

C Error Detector Comparison

As a supplementary experiment to Figure 4, in this section, we show the performance of different error detectors under the same average number of questions ("*target budget*"). Specifically, for each base semantic parser and each kind of error detector, we tune its decision threshold (i.e., p^* and s^*) such that the resulting average number of questions ("*actual budget*") is as close to the target as possible. In practice, we relax the actual budget to be within ± 0.015 of the target budget, which empirically leads to merely negligible variance. The results are shown in Table 9-10 for SQLNet and Table 11-12 for SQLova.

	SQLNet		
System	**Acc_{qm}**	**Acc_{ex}**	**Avg. #q**
no interaction	0.615	0.681	N/A
MISP-SQL$^{\text{Unlimit10}}$	0.932	0.948	7.445
MISP-SQL$^{\text{Unlimit3}}$	0.870	0.900	7.052
MISP-SQL$^{p^*=0.95}$	0.782	0.824	1.713
MISP-SQL$^{p^*=0.8}$	0.729	0.779	1.104
MISP-SQL$^{p^*=0.5}$	0.661	0.722	0.421
MISP-SQL$^{s^*=0.01}$	0.796	0.845	2.106
MISP-SQL$^{s^*=0.05}$	0.725	0.786	1.348
MISP-SQL$^{s^*=0.1}$	0.695	0.758	1.009
MISP-SQL$^{s^*=0.2}$	0.650	0.714	0.413

Table 6: Simulation evaluation of MISP-SQL (based on SQLNet) on WikiSQL Test set.

	SQLova		
System	**Acc_{qm}**	**Acc_{ex}**	**Avg. #q**
no interaction	0.797	0.853	N/A
MISP-SQL$^{\text{Unlimit10}}$	0.985	0.991	6.591
MISP-SQL$^{\text{Unlimit3}}$	0.955	0.974	6.515
MISP-SQL$^{p^*=0.95}$	0.912	0.939	0.773
MISP-SQL$^{p^*=0.8}$	0.880	0.914	0.488
MISP-SQL$^{p^*=0.5}$	0.835	0.879	0.209
MISP-SQL$^{s^*=0.01}$	0.913	0.942	0.893
MISP-SQL$^{s^*=0.03}$	0.866	0.912	0.515
MISP-SQL$^{s^*=0.05}$	0.840	0.892	0.333
MISP-SQL$^{s^*=0.07}$	0.825	0.880	0.216

Table 7: Simulation evaluation of MISP-SQL (based on SQLova) on WikiSQL Test set.

	[Lexicon]
	is greater than (or equivalent to)|equals to|is less than (or equivalent to)|does not equal to → OP[> (=) |= |< (=) |! =]
	is IN|is NOT IN|follows a pattern like|is between → OP[in|not in|like|between]
	sum of values in|average value in|number of|minimum value in|maximum value in → AGG[sum|avg|count|min|max]
	in descending order (and limited to top N)|in ascending order (and limited to top N) → ORDER[desc (limit N) |asc (limit N)]
	[Grammar]
(R1)	"col" in table "tab" → COL[col (table tab)]
(R2)	Does the system need to return information about COL[col] ? → Q[col‖SELECT $agg?$ col]
(R3)	Does the system need to return AGG[agg] COL[col] ? → Q[agg‖SELECT agg col]
(R4)	Does the system need to return a value after any mathematical calculations on COL[col] ? → Q[agg=None‖SELECT agg col]
(R5)	Does the system need to consider any conditions about COL[col] ? → Q[col‖WHERE col op val]
(R6)	The system considers the following condition: COL[col] OP[op] a given literal value. Is this condition correct? → Q[$terminal$‖WHERE col op $terminal$]
(R7)	The system considers the following condition: COL[col] OP[op] a value to be calculated. Is this condition correct? → Q[$root$‖WHERE col op $root$]
(R8)	Do the conditions about COL[col_i] and COL[col_j] hold at the same time? → Q[AND‖WHERE col_i .. AND col_j ..]
(R9)	Do the conditions about COL[col_i] and COL[col_j] hold alternatively? → Q[OR‖WHERE col_i .. OR col_j ..]
(R10)	Does the system need to group items in table tab based on COL[col] before doing any mathematical calculations? → Q[col‖GROUP BY col]
(R11)	Given that the system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, does the system need to consider any conditions about COL[col] ? → Q[col‖GROUP BY col^g HAVING $agg?$ col]
(R12)	Given that the system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, does the system need to consider any conditions about AGG[agg] COL[col] ? → Q[agg‖GROUP BY col^g HAVING agg col]
(R13)	Given that the system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, does the system need to consider a value after any mathematical calculations on COL[col] ? → Q[agg=None‖GROUP BY col^g HAVING agg col]
(R14)	The system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, then considers the following condition: COL[col] OP[op] a value. Is this condition correct? → Q[op‖GROUP BY col^g HAVING $agg?$ col op val]
(R15)	Given that the system groups items in table tab^g based on COL[col^g] before doing any mathematical calculations, does it need to consider any conditions? → Q[NONE_HAVING ‖GROUP BY col^g NONE_HAVING]
(R16)	Does the system need to order results based on COL[col] ? → Q[col‖ORDER BY $agg?$ col]
(R17)	Does the system need to order results based on AGG[agg] COL[col] ? → Q[agg‖ORDER BY agg col]
(R18)	Does the system need to order results based on a value after any mathematical calculations on COL[col] ? → Q[agg=None‖ORDER BY agg col]
(R19)	Given that the system orders the results based on (AGG[agg]) COL[col], does it need to be ORDER[od] ? → Q[od‖ORDER BY $agg?$ col od]

Table 8: Extended lexicon and grammar for MISP-SQL NLG module to handle complex SQL on Spider.

Avg. #q	Probability-based		Dropout-based	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
0.5	0.672	0.732	0.663	0.726
1.0	0.725	0.775	0.706	0.765
1.5	0.778	0.820	0.749	0.809
2.0	0.812	0.848	0.796	0.845

Table 9: Comparison of error detectors for SQLNet with a target average number of questions on WikiSQL Dev set.

Avg. #q	Probability-based		Dropout-based	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
0.5	0.669	0.729	0.656	0.720
1.0	0.722	0.773	0.695	0.758
1.5	0.765	0.810	0.740	0.801
2.0	0.805	0.844	0.790	0.842

Table 10: Comparison of error detectors for SQLNet with a target average number of questions on WikiSQL Test set.

Avg. #q	Probability-based		Dropout-based	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
0.2	0.844	0.885	0.829	0.881
0.4	0.876	0.910	0.856	0.905
0.6	0.902	0.932	0.887	0.927
0.8	0.921	0.947	0.913	0.941

Table 11: Comparison of error detectors for SQLova with a target average number of questions on WikiSQL Dev set.

Avg. #q	Probability-based		Dropout-based	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
0.2	0.832	0.877	0.823	0.878
0.4	0.865	0.902	0.851	0.901
0.6	0.895	0.926	0.881	0.922
0.8	0.915	0.941	0.904	0.936

Table 12: Comparison of error detectors for SQLova with a target average number of questions on WikiSQL Test set.