Extending MISP

Aiswarya Bokil **Akshay Kekuda** **Ayyappa Kolli**

Abstract

This project is about 'Extending Model based Interactive Semantic Parsing' to other base parsers and the challenges faced while implementing the same and building an interface to interact with the model. We also discuss the motivation, efforts and design in building an interface to interact with the Model Based Interactive Semantic Parsing model.

1 Introduction

To reduce the ambiguity of natural language occurrences and further boost the accuracy of base semantic parsers for Text-to-SQL conversion, an interactive model called Model based Interactive Semantic Parsing has been suggested. In this project we tried integrating multiple base semantic parsers and created an interface to interact with MISP in a convenient way.

We chose **'BRIDGE – Tabular Semantic Parser' and 'IRnet for complex and cross-domain Text-to-SQL.'** on Spider dataset to integrate with MISP. Implemented a web interface to interact with MISP which makes the interaction better with clear view of the table and ensures no error in the human interaction. This feature extends the functionality of MISP for conversion of Natural Language Text to SQL text.

2 Base parser Integration

2.1 MISP understanding

MISP can improve on base parser performance and can increase the interpretability of the base parsers. It can help save annotation time and can also, in some cases, reduce the amount of training data required. The MISP contains 3 main components, - World Model - Error Detector - Actuator The world model enables user interaction, records it, and transitions to a new agent state. Error detector interprets this state to decide when human intervention is required using either probability-based or dropout-based evaluation methods. The Actuator is a Rule-based Natural language question generator that uses seed lexicons. The 2 cornerstones for MISP implementation include creating Hypothesis and semantic unit. Each of us attempted to understand the workings of this architecture by implementing it with the EditSQL and SQLova base parsers.

2.2 Base parsers

We experimented with 5 base parsers: spider-schema-gnn, RATSQL, SLSQL, Tabular Semantic Parser and IRNet. Of these, spider-schema-gnn, RATSQL and SLSQL had deprecated library issues. In addition to incompatible libraries, RATSQL built their model on a docker and a docker image was not provided to run their model. Only Tabular Semantic Parser(BRIDGE) and IRNet worked for us and so we show our results for these two.

BRIDGE is a powerful sequential architecture that contextualizes question and schema representation. It uses pre-trained BERT model to increase generalization and thereby increasing parsing accuracy. Tabular semantic parser focuses on building executable output. The model, hence, employs decoding and post-processing steps built on schema to ensure executability. The database schema and natural language utterance are combined to build a tagged sequence that is fed to the BRIDGE model to output program sequences along with the associated tagged sequence's probabilities. The syntactical correctness is verified by an SQL checker.

IRNet employs a neural approach to tackle the mismatch problem with intermediate representation and schema linking. It breaks down the synthesis process into three phases. First, it performs a schema linkage over natural language question and the database schema. A neural model then

synthesizes an intermediate representation between natural language question and SemQL, a domain specific language. At the last stage, IRNet performs inference using SemQL and domain knowledge.

2.3 Integration challenges

We found it challenging to integrate both IRNET and BRIDGE mainly due to the 2 layered decoding steps they implement.

For BRIDGE, we implemented additional methods in the decoder to ensure it could return the semantic units required for it to interact with the MISP agent. The semantic units generated after we had inherited the MIPS's semantic units and hypothesis class failed the SQL checker. The tabular semantic parser eliminates the sequences that do not pass the checker and, due to the change to accommodate semantic units, none of the tags generated got through. To circumvent this, we understand that we might have to re-modify the SQL checker to make sure it picks the decoded sequences from the generated hypothesis, join them and then run the checker instead.

For IRNet, integrating the semantic unit representation and modifying the decode function of the base parser to return an instance of the Hypotheses class was challenging. This was especially difficult since the neural model required a SemQL representation. The decode method of IRNet returns a SemQL, which is a tree like structure. In order to generate Semantic units of MISP, we had to modify this SemQL hypothesis. But the SemQL hypothesis did not contain information about the clauses like GROUPBY, HAVING, FROM. These SemQL hypothesis units were not enough to generate the hypothesis unit of MISP which is needed to do any sort of error correction. IRNet generates the entire SQL query by deterministically infering a SQL query from the synthesized SemQL query with domain knowledge. This kind of approach does not give scope for doing a token by token decoding which is what a MISP base parser needs.

2.4 Methodology for integrating base parser to MISP

The after-mentioned points outline the main steps one will have to undertake to execute any base parser integration with MISP. Agent: To have an agent help, we can directly implement the agent from the MISP architecture. We might have to add additional methods for evaluation, but the functionality for calling user interaction, gold user simulation, question and option generation and decoding is all mentioned in the MISP code. MISP SQL error detector: We directly inherited the error detector code without any further alterations to this file. MISP SQL world model: The world model, which contains the definitions for decoding, needs to be modified to accommodate the base parser's decode methods. We simply call the base parser's decode method for this and check the passes to decide when to return the hypotheses. We also define the apply positive feedback and apply negative feedback methods. The definitions are not given in the base parser and will need to be implemented during integration with the base parser. To implement the world model.py, the decode methods for the base parsers needed to be modified to ensure that it returns a hypothesis. To do so, we inherited the hypotheses class in the base parser and re-wrote the decode method To build the 'tag seq' (part of hypotheses), we need semantic units and hence we inherited the semantic units from the MISP's utils class. This had to be customized to suit different base parsers.

3 Interface for MISP (MISPBot)

Real user interaction with MISP is when users can communicate with the model and enhance the accuracy of the model in conversion of text to SQL queries. We developed a chat bot using web interface, where users can answer the questions asked by bot, view the table information required for answering the question. This will enable the users understand the behaviour of MISP.

3.1 Need for interface

In existing MISP the real user interaction is through terminal, the model takes an input from the evaluation set and post the question, shows the table definition and waits for user input. Users have to scroll and look at the table for each question and as the interaction increases it is inconvenient to scroll and look at the table.

3.2 Methodology and Design

The design of this project includes the interaction with agent of MISP. We created a backend to maintain the current state of the model, post questions to user, and take the response from user and posting the SQL query.

In order to integrate MISP with a Web Based Interactive Agent, we create a InteractiveWebAgent

class that implements all the methods of the Base MISP agent. In order for the InteractiveWebAgent to work, we need to maintain states of the MISP model. Based on the type of response the User sends and based on the working of MISP model, we define 6 states for MISPBot. Interaction in MISP with real user starts with MISPBot initializing the model and starting the real user interactive session. This forms the initialization state State 1 of MISPBot. MISPBot then transitions to State2. Here, when the user is ready, the MISPBot displays the NL Query and the table corresponding to it. Then MISPBot transitions to state 3, where it checks for the presence of erroneous semantic unit. If an error exists MISPBot shows the erroneous semantic unit, and the question from the question generator about the erroneous semantic unit is shown to the user. Otherwise, MISPBot goes back to the State 2, the NL Question State. State 3 is the following: based on the feedback from the user, the MISPBot applies positive/negative feedback. When a positive feedback is given, the MISPBot transitions to State 2, where the next erroneous unit is shown. On the other hand, when a negative feedback is given, MISPBot transitions to state 4. In state 4, MISPBot prepares clarification options to be shown to the user. When the user selects an option, MISPBot transitions to state 5, where it incorporates this selection, and transitions to State 3, where it generates the next erroneous semantic unit and the question corrbesponding to it. This cycle continues and when there are no more erroneous semantic units MISPBot transitions back to state 2 to show new NL Query.

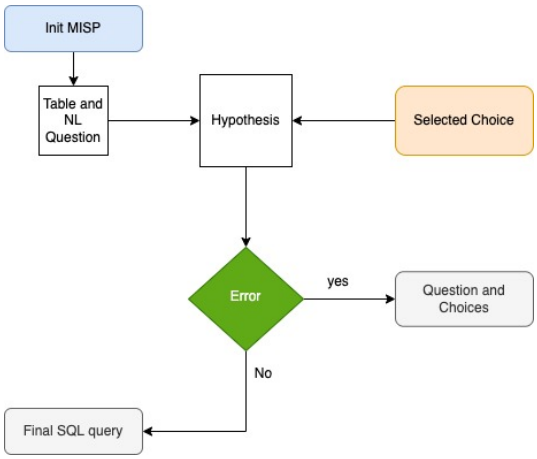

Figure 1: Archtitecture

In the UI users can see a chat bot and the table which he is talking about. In the chat bot section, users can see the questions, see the options for the questions and also write their responses. The major challenge with creating an interface is the model needs to be deployed in a server because the current MISP only supports few system and environment configurations, so we need a container to host it. And cross origin requests have to be allowed if multiple hops happen in network.

The architecture we followed is creating a react app for UI, Flask app to host the model in server. For communicating with the backend, APIs are exposed in server, and these are hit from UI when needed. Each API executes an agents functionality like giving feedback, initiating states.

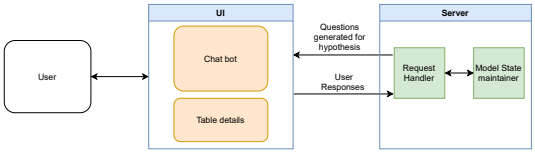

Figure 2: Archtitecture

code:https://github.com/ayyappa7/interactive-SQLova

4 Results

We are able to build the model and abel to see the following results. here are the images of tabel and chatbox visible in UI.

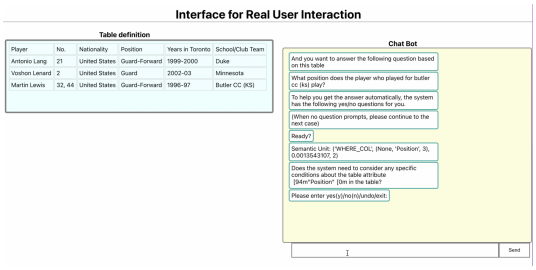

Figure 3: Initial state

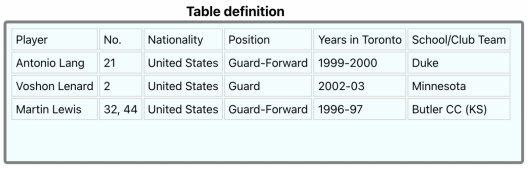

Figure 4: Table view

5 Contributions

The first part of the project involved understanding the MISP code and its replication on EditSQL and

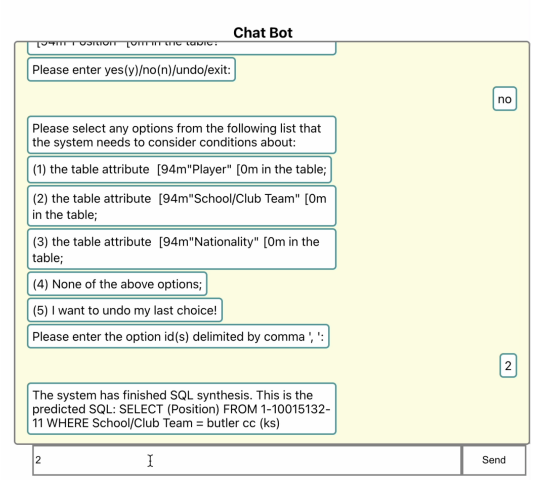

Figure 5: Chat Box

SQLova. Aishwarya Bokil provided the paper summary and presented implementation details about MISP. Akshay Kekuda worked on understanding SQLova and replicating it's results while Ayyappa Kolli did the same for EditSQL. Ayyappa Kolli and Akshay Kekuda, experimented with setting up and spider-schema-gnn, RATSQL, SLSQL, Tabular Semantic Parser and IRNet. Aishwarya Bokil worked on implementing Tabular semantic parser and extending the MISP implementation to this base parser. Akshay Kekuda worked on comprehending the functioning of IRNet and integrating MISP into this parser to further boost its accuracy. For MISPBot, Akshay Kekuda designed the InteractiveWebAgent for MISPBot. Ayyappa Kolli worked on building the MISPBot architecture for the front end and setting up server in OSC and exposing APIs that would allow users to interact with chosen base parsers.

6 Conclusion

Integrating base parsers with MISP is a challenging task. One needs to ensure that the base parser's decode methods support returning the hypothesis structure. Our experiments with MISPBot has shown promising results in enhancing user interaction with MISP. We were successfully able to port real user interaction from a terminal to a UI-based interaction MISPBot which is visually more appealing. As we discuss in the future work section, MISPBot can be extended to incorporate more advanced features, which would be difficult to achieve with the current setup.

7 Future Work

The web based UI can handle more features like taking users actual question and identifying the table realted to it. We can also add the Options to train/test the MISP model can also be visualized. Support for User Query can be added. Integrate ASR to take users voice input. By putting more work in maintaining intermediate results and showing them to the user, we can make the model explainable.

8 References

Ziyu Yao , Yu Su , Huan Sun and Wen-tau Yih. 2019. Model-based Interactive Semantic Parsing: *A Unified Framework and A Text-to-SQL Case Study*

Ziyu Yao, Yu Su, Huan Sun, Wen-tau Yih. 2020. *An Imitation Game for Learning Semantic Parsers from User Interaction*

Wonseok Hwang, Jinyeong Yim, Seunghyun Park and Minjoon Seo. 2019. *A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization*

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*

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu and Dongmei Zhang. 2019. *Editing-Based SQL Query Generation for Cross-Domain Context-Dependent Questions*

Xi Victoria Lin, Richard Socher and Caiming Xiong. 2019. *Bridging Textual and Tabular Datafor Cross-Domain Text-to-SQL Semantic Parsing*