Improving the User Experience on MISP

Noah LaPolt, Shlok Sah

Ohio State University, Department of Computer Science and Engineering

Abstract

MISP works such that it has an agent state with three major components – World Model, Error Detector and Actuator. The World Model calls the base semantic parser and represents the user intent. The Error detector checks the metadata of SQL component and determines based on probability to ask user for more clarification. The actuator generates the corresponding natural language question to the user to gather more data about the asked question. The problems in current approach are – limited question base, absence of GUI and lower business implications current SQL parser as it works for English speaking users. Also, building a multilingual semantic parser is computationally expensive. With these problems in mind, a question paraphraser and translator linked to the Actuator of MISP model was designed. The GUI interface designed is linked with the User environment that supports the input and output utterances to/from user. The USP of the improvement is that it is a simple plug and component which can be linked to any semantic parser with an actuator.

1 Introduction

MISP Edit SQL creates SQL queries from questions inputted in natural language via a model decoder pair. It also can generate questions to help clarify what data it should be collecting from the database. However, interactions with the model are robotic and clunky. In this paper possible solutions are explored and implement in order to improve the actuator and user interactions with the model.

The first step was to improve the actuator in MISP Edit SQL semantic parser by paraphrasing hard coded questions generated by the actuator. A text-to-text transformer was fine-tuned allowing it to perform the paraphrasing tasks. This change can help in improving the use-case of the semantic parser and letting it be implemented in real world scenarios where any kind of question from the user can be expected, like Alexa and Siri. Keeping the conversations with the model as "human" as possible avoiding questions and answers that simply regurgitate what is missing or found. For example, say the model was asked, "What is the name of the largest country?" When responding, the model should not say, "Which attribute should sort the sizes of the country?", but instead respond "Should I sort the countries by population or some other quantity?" In this instance it initially offers a suggestion as opposed to simply asking for more information, but still allows for any other quantity to be used to sort the data. The final objective would be to talk "with" the model not "at" the model.

The next step was to implement a graphical user interface (GUI) for the user to interact with the model directly. Currently, MISP is run from the command line and utilizes already existing questions stored withing a json file. Upon completion all questions, interactions, and response are printed to a text file. As such there is no new user interactions unless they are added into the json file with correct formatting and representation. A GUI was implemented with the

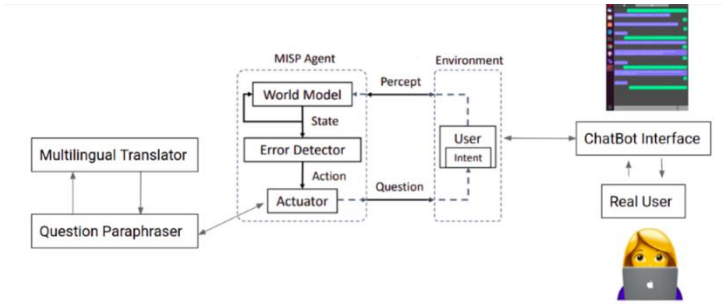

Figure 1 Graphical representation of model

idea of allowing for the user to directly make SQL request via a natural language question and for the results to be displayed. The resulting GUI was meant to simulate talking to another person on a messaging app.

Finally, a translator has also been implemented within this framework to allow interaction with multilingual user base. This is beneficial for improving the use case of the semantic parser and allows it to be used by larger customer base. It does not require a use of multilingual semantic parser and the actuator translates and feeds the parser "English" utterances. Displaying the users selected language preference within the GUI. Overall, these changes are meant to improve the user experience and benefit the business by reaching a larger user base.

2 GUI

For user ease of use a GUI was created which connected to the parser. The GUI allowed for natural language questions to be sent to the model. The natural language questions could be sent in a variety of different languages and could be displayed in a variety of different colors to assist with color blind individuals and add more depth to the page.

2.1 Framework

In order to make the GUI easy to access and to allow for multiple people to use the model at the same time it was served as a webapp. The webapp utilized a python library called Flexx that lets python code be the backend for a website. With a python backend the model could be created and accessed just as it was in test__with_interaction.sh. However, the function interaction in interaction_editsql.py needed to be edited in order to do a single interaction at a time as opposed to the entire dataset of interactions. Once the script had finished building the model it was served to the local host allowing for multiple users to interact with it at the same time without it running on their machine. However, the resulting load of multiple models running at the same time on a regular desktop led to multi-client support being disabled during testing.

Originally, new questions were meant to be entered in by the user and response were meant to be created based on the new natural language inputs. While exploring the model it was discovered that new questions would have to have SQL tokens. They were used when a new hypothesis was created utilizing the agents interactive_parsing_session function. Removing the true SQL resulted in no questions being asked as the user object could no longer evaluate the correctness of the SQL generated. An attempt at a solution was to create a custom user, agent and model interface that did not rely on the true SQL tokens. Due to the time constraints the idea was abandoned in order to focus on finishing the design with the already existing questions from the given database.

2.2 Design

The GUI was designed with the intention of getting users to come back to use it again. In order to do this a colorful, organized and responsive GUI was utilized. The decision to add color was based on the findings that blue helped calm most viewers and green was associated with nature (Rider. 2009). To make the model seem more natural and easier to talk to. With the addition of colors came the addition of color blind accessibility.

There are seven different types of color blindness that are currently documented. Each of these different types of color blindness see the normal spectrum of light differently. In order to test colors that would be just as effective for color blind users as other users different a color blindness color generator was utilized (Bianchi).

This generator shows what a color looks like normally versus how specific different types of color blindness see the color. When a different color blind option was selected the page would reload with that specific color theme.

2.3 Features

The features of the GUI were a language button, a color blindness button, a chat box, and a data display. Both buttons dropped down menus that could be scrolled through to find options that the user may want. The chat box simply showed a history of past messages sent by the model and user. The data display shows a cluster of circles that disappear if a table needs to be show. The entire GUI was also resizable and adjustable to show more chat or more data if the user desired. All together the parts created the interactive web interface.

2.4 Results

The final GUI used could correctly interact with the model and display data, in the correct color and language. Although new questions could not be sent to the model on the fly with more time it would be possible to send the new questions directly into the predictor. In a few cases the resulting table that is displayed in the data section is too big to be seen all at once. Also, the text sent to the GUI has the chance to be in the incorrect order if the webpage has trouble keeping up.

3 Question Paraphraser

Under the T5 framework, a transformer was fine-tuned for the paraphrasing task. T5 is state of the art transformer model developed by Google which is unified into text-to-text format in contrast to BERT style models that can output either a class or a span of the input. It can achieve good benchmarks on NLP tasks while being flexible to be fine-tuned on wide variety if downstream tasks. T5 was trained with the original sentence as the input and paraphrased sentence as the output.

3.1 Datasets

The Quora question pair dataset was used. Which was released by Quora to tackle their important product principle that there should be a single question page for each logical question as a simple example, the queries “What is the most populous state in the USA?” and “Which state in the United States has the most people?” should not exist separately on Quora because the intent behind both is identical.

The second dataset used is PAWS which was released by Google in 2019 containing 108k human-labelled and 656k noisily labelled pairs. The Wikipedia version (PAWS Wiki) was used to fine tune the T5 model. The main motivation behind releasing this dataset was that existing paraphrase identification datasets lacked sentence pairs that have high lexical overlap without being paraphrases. Models trained on such data failed to distinguish pairs like flights from New York to Florida and flights from Florida to New York. Cross database fine tuning was performed using QQP and PAWS Wiki datasets.

3.2 Fine-tuning

T5 allows us to use same model, loss functions, hyperparameters across a diverse set of tasks. The ‘Huggingface’ T5 paraphrase transformer was modified to perform fine tuning on both Quora and PAWS dataset. Several experiments were performed with different Optimizers, learning rates and epochs but the original values of hyper parameters performed the best. The training was performed on the Ohio State Supercomputer with GPU acceleration. A total of 2 epochs took around 10 hours of training time. The key is the format of the input and output given to the T5 model trainer. For any given question pair from the dataset, the input (source) and output (target) were given to the T5 model as shown below –

```
paraphrase: What are the ingredients required to make a perfect cake? </s>
```

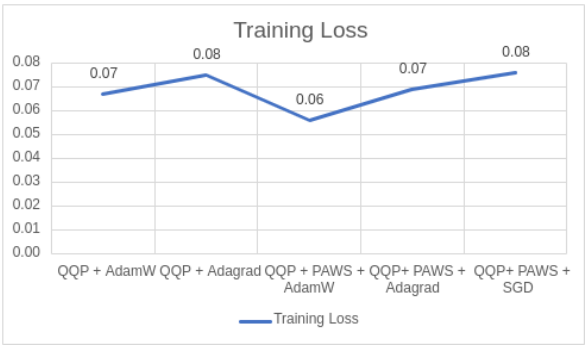

Figure 2 Training loss of different methods

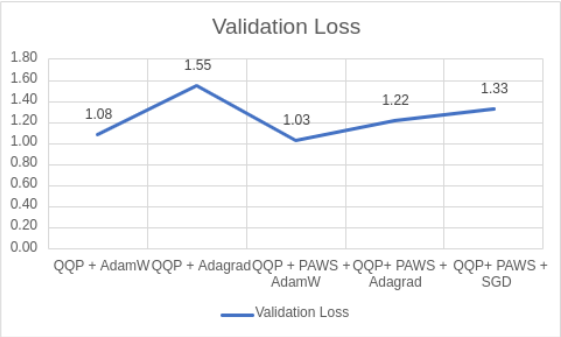

Figure 3 Validation loss of different methods

4 Experiments

The proposed question paraphrasers was evaluated with combination of Adagrad, AdamW and SGD optimizers with different learning rates along with QQP and PAWS datasets.

The initial setup was to fine tune T5 transformer with original configuration and Quora dataset which gives us a validation loss of 1.08. The process was repeated after appending the pre-processed PAWS Wiki dataset with the Quora pairs dataset to get a more "natural" paraphrasing rather than technical jargons (typical in a Quora question). After running, 2 epochs a validation loss of 1.03 was achieved which is better, but not a significant improvement.

Next, experiments between AdamW, AdaGrad and SGD optimizers were done for the fine-tuning task it was found that AdamW performing the best by giving a loss of 1.03 for PAWS+ Quora dataset combination.

As stated earlier new dataset did not improve much in terms of loss, but there was a difference of paraphrasing of different sentences. For fine tuning the T5 model, OSC (Ohio Supercomputer Center) Owens supercomputer with GPU acceleration was used.

5 Multilingual Translator

A multilingual translator was implemented from deep-translator library which allows us to interact with users with different lingual background. It was developed by Nidhal Baccouri to provide free and unlimited translation capabilities using famous online translators like Google translate, Microsoft translate, and Papago translate. This increases the usability of the parser as it can support different languages.

The question paraphraser uses Google translator module of deep translator library. How it works is that the user is asked for the language that they would like to interact in and then, the questions are translated into English which on which the Edit SQL parser is based on. Same idea is applied to display output from the question generator into the desired language. The basic idea is to keep the SQL parser unaware of the translation taking place outside its scope. This makes it easier to work with English text to SQL parser like Edit SQL without having to train it on different languages.

Original Sentence	Original Model (QQP+AdamW)	New Model (QQP+PAWS+AdamW)
Can I consider any specific conditions about cities?	How do I consider specific conditions about cities	What are some conditions to consider when doing research about cities?
Shall I order the results based on game scores?	What are some way to sort the results on Google based on game scores?	When the system takes a match scores value, then how does one order games out?
Please confirm if you need information on flight travel time?	If I need info about flight travel time, please confirm if I need it	If you want more information on time of flight please confirm this
Select any options below that I need to consider	What options are listed below that you need to consider when seeking advice for a career	Is there any option there that I need to consider below

Table 1 Comparison between original and new sentences

6 Conclusion and Future Work

Through the use GUI and a question parser a more interactive and natural experience was created. The GUI utilized specific colors to catch the eye and offered color blindness options for those that could not see the colors. The question paraphraser created more natural questions for the user. Both the GUI and question paraphraser can be

improved with more time, computation power and research.

Going forward, further exploration into implementation of a more in-depth GUI and more accurate and custom question paraphraser are to be considered. The GUI should allow for new natural language questions to be asked. As well as utilizing different SQL models per situation.

For the question paraphraser, a different approach entirely could be used, such as Pegasus. As discussed with fellow classmates, the performance of different user languages could be evaluated. Some best practices like Adapters for T5 transformer and Gradual unfreezing can be part of the experiments. Finally, testing the GUI and paraphraser on a different model like SQLova could lead to better results.

Contributions

Both authors had equal contribution. Noah helped set up the MISP system and worked on the GUI. Shlok worked on the question paraphraser and setting up the fine-tuning tasks on the supercomputer. This research was a collaborative effort, and both help each throughout the entire process. Both authors completed the section of the paper that they implemented.

Note: Our source code is available here: https://github.com/shloksah/MISP

Acknowledgments

We would like to thank Ohio State Supercomputer Center for allowing us to use their computer resources to help in this research. We would also like to thank our Professor Huan Sun and our course TA Ron Chen for solving our doubts and guiding us with the right approach to tackle those problems.

References

Adam Roberts and Colin Raffel. 2020. Exploring Transfer Learning with T5: the Text-To-Text Transfer Transformer. Google Research

Fabrizio Bianchi. Coolors.

Nidhaloff. 2021. Nidhaloff/deep-translator. GitHub.

Ramsri Golla 2020. Ramsrigouthamg/t5_paraphraser. Hugging Face.

Rose Rider. 2009. Color Psychology and Graphic Design Applications. *Senior Honors Theses. 111. Liberty University. Lynchburg, Virginia.*

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First Quora Dataset Release: Question Pairs Quora.

Suraj Patil. 2020. Google colaboratory t5_fine-tunning. Google Colab.

Yao Z., Tang Y., Yih W.-tau, Sun H., & Su Y. 2020. An imitation game for learning semantic parsers from user interaction. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Yuan Zhang. 2019. Paraphrase Adversaries from Word Scrambling. Google-Reserch-Datasets GitHub.

Appendix A. GUI

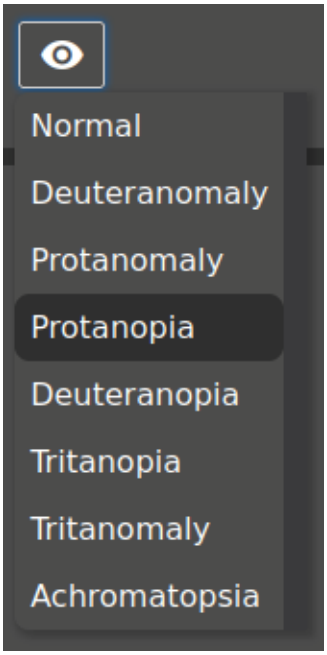

Figure 5 Color blind options

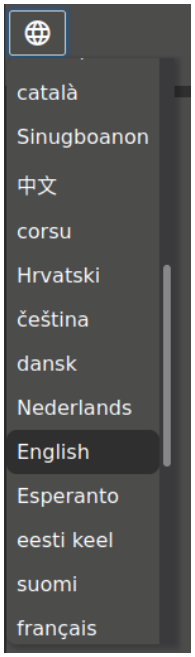

Figure 4 Language options