An Empirical Study of Model Errors and Users' Error Discovery and Repair Strategies in Natural Language Interfaces for Database Queries

ANONYMOUS

The natural language to SQL (NL2SQL) models can help users who are not experts in the programming language more easily access digital databases by automatically translating the natural language input into SQL queries that can be understood by computer. However, current state-of-art models only reach 70% accuracy in popular benchmark datasets, the difficulty for users to identify and repair errors limits their wide adoption. In order to address this problem, we build the first comprehensive taxonomy of NL2SQL errors as well as statistical analysis based on the result of three state-of-the-art models on the Spider dataset[6], which is the benchmark dataset used for training and evaluating these models. Then we conduct a controlled user study that compares the effectiveness and efficiency of interaction strategies for error discovery and repair. Based on the findings, we discuss design implications for natural language interfaces for database queries and future research directions.

CCS Concepts: • **Human-centered computing → Empirical studies in interaction design**.

Additional Key Words and Phrases: empirical study, human computer interaction, database systems

ACM Reference Format:
Anonymous. 2018. An Empirical Study of Model Errors and Users' Error Discovery and Repair Strategies in Natural Language Interfaces for Database Queries. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

TL: some generic stuff on why NLQ is useful

TL: popularity of NLQ in ML/NLP community, evidence, rising industry interest and community interest due to its strong potential in useful applications—most work focus on improving the accuracy but few looked at the errors in NLQ

TL: many ML models are proposed, objective accuracy measures are reported

TL: why errors in NLQ are problematic – while many models have high accuracy (70%+), not practical/useful if end users cannot easily identify errors made by the models and repair them. cite work in human-ai colab

TL: lack of knowledge on WHAT errors the current models make

TL: some novel interface design (DIY), interaction strategies etc. proposed for helping users handle errors in NL DB queries and other NL queries in general. but unclear (1) what types of errors are they useful for (2) (3)

To bridge these gaps in the prior literature, this paper presents two studies: The goal of the first study is to understand the characteristics of errors made by current models for understanding natural language database queries. We proposed a rich taxonomy of NL2SQL errors by conducting an iterative coding analysis on a large-scale dataset that contains flawed SQL queries generated by leading NL2SQL models. The study also produced an annotated dataset of errors types in natural language database queries.

Built upon the results from the first study, we conducted a second study that focuses on the user performance and behaviors in handling errors in natural language database queries. Through a controlled study with X participants, we tested the effectiveness of XXX on discovering, identifying, and repairing errors using the dataset we created in the first study. The results suggest that…

TL: dicuss design implications and impacts

In summary, this paper presents the following four contributions:

(1) We built the first annotated dataset of errors and error types for NL2SQL models.
(2) We created a taxonomy of error types through open and axial coding procedures.
(3) We did a controlled user study that measures the effectiveness of different interaction strategies for discovering, identifying, and fixing errors in natural language database queries.
(4) We discuss the design implications for error handling in natural language query interfaces

2 RELATED WORK

TL: Prior systems/models in NL2SQL

TL: Prior work in error discovery and repair in human-ai systems: e.g., chatbots, interactive infoviz tools, human-ai co-creation

TL: Prior work in studies for understanding errors/breakdowns in human-ai interaction

3 STUDY 1

In the first study, we annotated the types of errors made by three state-of-art NL2SQL models on Spider and proposed a taxonomy of understanding errors in NL2SQL models using a grounded theory approach.

Specifically, the study seeks to answer the following research questions:

		easy	*medium*	*hard*	*extra hard*
original Spider	*train*	1983	2999	1921	1755
	dev	248	446	174	166
re-split Spider	*train*	1604	2363	1516	0
	dev	627	1082	579	0

Table 1. Stats of the original Spider dataset and our re-split one

Model	# Errors	Accuracy	Orig. Accu.
SmBoP + GraPPa [4]	465	85.0%	71.1%
BRIDGE v2 + BERT(ensemble) [2]	850	62.8%	68.3%
GAZP[8]	1070	53.2%	53.5%

Table 2. Descriptive statistics and the accuracy of each model we reproduced

RQ1: What types of errors do existing NL2SQL models make?
RQ2: Which part of the generated SQL Query is most error-prone?
RQ3: How error types vary in various models?
RQ4: How do each model perform differently in a query?

3.1 Dataset Preparation

We built the NL2SQL error dataset based on the Spider dataset [7], which is a large-scale human-labeled dataset for complex and cross-domain natural language database queries. The whole Spider dataset consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables covering 138 different domains. This dataset is commonly used as a gold standard metric for evaluating NL2SQL models—as of August 2021, at least 70 models[1] have been trained and evaluated on Spider. Spider is a challenging dataset as it tests models' capabilities on handling complex (e.g., multiple JOIN, nested queries, complex filters and conditions) and cross domain (i.e., the testing dataset contains tables in previously unseen domains with unique database schema) queries.

As the original testing set is kept unseen for unbiased evaluation, we re-split the original training and developing set into 5483 training samples and 2288 developing samples. The databases used in both sets are non-overlapping. As all the recent SOTAs [4][8][2][5] report unsatisfying performance on extra hard samples, we exclude those samples in our experiments. The stats of the new dataset are reported in Table X

We selected three representative models in our study: SmBop[4], BRIDGE[2] and GAZP[8]. The reasons are four-folds: (1) The code for each model is officially open-sourced. (2) All the models rank top 10 in the official leading board of the Spider dataset as we conducted the experiments as we conducted the experiment. (3) These models are evaluated on execution accuracy with actual value, rather than comparing the similarity between the ground truth and generated hypothesis. (4) All the three models are built upon distinctive approaches which cover the main-stream solutions for today's NL2SQL problem.

Table X shows the descriptive statistics and the accuracy of these models after training on the re-split Spider dataset. We also report each model's original reported accuracy on the unreleased test set for comparison.

[1]https://yale-lily.github.io/spider

3.2 Annotation Methods

3.2.1 Building Taxonomy. Ground theory approach We randomly separated all the errors in the dataset into 7 batches. Coders with domain expertise in SQL annotated the starting four batches. In the first batch, all our coders were asked to write open-ended descriptions independently to each of the error. After finishing this step, the result from all coders were put together to discuss the corresponding error type and category. During this process, descriptions that had distinct meaning would be labelled by different upper case letter, while those that were similar in category but different in minor would be indexed by a category label plus a varied suffix. For example, as letter "A" denotes *"Keyword error in SQL query"*, through discussion, all the coders agreed that *"Miss a keyword in the SQL query"* belongs to error category "A", so that statement is labelled by "A1". The label can also denote inter-category overlap. For example, "AN2" means the error type belongs to both category "A" & "N".

Iteratively refine the codebook The preliminary version of the codebook is iteratively refined for the following batches. During this process, the coders keep annotating the data using the updated codebook from the last batch independently, if they encounter any error types that are not recorded in the codebook, they will write open-ended descriptions for discussion later. The new code will be added to the codebook if all the coders make an agreement.

The final version of our codebook contains 13 error categories, inside each category contains error types that point to specific errors. The codebook is reported in table xxx.

3.2.2 Coding Procedure. To start with, there are four coders in total to do the annotation. We calculate Fleiss' Kappa, Krippendorff's Alpha[3] and stability score to measure the inter-rater reliability.

The Fleiss' kappa is used for measuring the reliability of agreements between coders' annotation. However as it requires each coder to assign only one error type to a query to make sure the number of codes for each query is a constant, we cannot use it directly to our case where a certain query has a variable number of codes. In order to address that, for each of the unique code in a query, we consider how many coders have selected the exact code. Therefore, we can work out the score for each code in a query. At the end of each batch, we average all of the scores to be the Fleiss' kappa for this round of annotation.

Similar approach is applied on computing Krippendorff's alpha score, besides we adopt method in [1] to address the problem that K's alpha can only measure agreement of no more than two annotators.

ZN: Illustration to stability score

As the agreement score stabilized after batch #4, only two coders took part in the annotation for the remaining batches. After annotating the whole dataset. The two coders made an additional review to all the annotation results in the previous batches and recode using the newest codebook.

The average scores for Fleiss' kappa among all batches is xxx, Krippendorff's alpha is xxx and stability score is xxx

The stability score is to measure the degree of agreement among annotators that a number of SQL queries have a particular type of error.

3.3 Annotation Interface

TL: include a short description and a screenshot of the interface

The error annotation interface is showed in fig.xxx. It consists of three components: (1) A natural language query

Error category	Error type	Description
A: Keyword error in SQL query	A1	Miss a keyword in the SQL query
	A1a	Miss NOT keyword
	KA1b	Miss a ORDER BY keyword
	A1b	Miss a DISTINCT keyword
	A1c	Miss an ALL keyword
	A1d	Miss an EXCEPT keyword
	A1e	Miss UNION keyword
	A1f	Miss a HAVING keyword
	A1g	Miss INTERSECT
	A2a	Redundant DISTINCT
	A2b	Redundant UNION
	A2c	Redundant EXCEPT
	AN2	Miss a GROUP BY keyword in the SQL query
B: Column error	BL1	Return a wrong column in SELECT
	BL2	Return a redundant column in SELECT
	BL3	Miss returning a column in SELECT
	BF4	Use a wrong column in WHERE
	BE5	Use a wrong column in JOIN
	BN1	Use a wrong column in GROUP BY
	BK1	Use a wrong column in ORDER BY
C: Table error	CE2	Miss a table to JOIN
	CE3	JOIN the wrong table
	CF5	Use a wrong table in WHERE
	CL1	Use a wrong table in SELECT
D: Value error in WHERE	D1	Wrong value in WHERE clause
	D1a	Value case error in WHERE clause
	D1b	Plularity error in WHERE clause
	D1c	Synonym error in WHERE clause
E: Join error (JOIN)	CE2	Miss a table to JOIN
	CE3	JOIN the wrong table
	E1	Redudant JOIN
	BE5	Use a wrong column in JOIN
F: Condition error (WHERE)	F1	Wrong comparator (, , =, !=, etc)
	F2	Wrong boolean operator (AND, OR etc.)
	F3	Redundant condition
	F4	Missing condition
	BF4	Use a wrong column in WHERE
K: Sorting error (ORDER BY)	KA1b	Miss a ORDER BY keyword in the SQL query
	K2	Wrong sorting direction
	K3	Use a wrong condition to sort
	K4	Redundant sorting
	K5	Miss sorting a column
L: Select error (SELECT)	BL1	Return a wrong column in SELECT
	BL2	Return a redundant column in SELECT
	BL3	Miss returning a column in SELECT
	CL1	Use a wrong table in SELECT
	L4	Use wrong aggregation function
	L5	Miss aggregation function
N: Group error (GROUP BY)	BN1	Use a wrong column in GROUP BY
	AN2	Miss a GROUP BY keyword in the SQL query
	N3	Redundant GROUP BY
O: HAVING	O1	Miss HAVING pattern
	O2	Redundant HAVING
	O3	Wrong condition in HAVING
P: LIKE	P1	Miss LIKE pattern
	P2	Redundant LIKE pattern
	P3	Wrong LIKE pattern
Q: LIMIT	Q1	Redundant LIMIT
	Q2	Miss LIMIT
	Q3	Wrong LIMIT clause
R: INTERSECT	R1	Redundant INTERSECT

Table 3. Error taxonomy for NL2SQL models

and the corresponding ground truth and hypothetical SQL query are displayed in A. (2) In error annotation sector, the annotator first decides which part of the SQL is wrong in (B), after that, the annotator is supposed to choose error types from the checkbox in (C). If the error type is not included in (C), the system provides a input box for writing open feedback. (3) The in-used table is displayed on the right side of the canvas to help annotator identify the error types.

3.4 Results (Data Analysis)

ZN: Bar chart for counting errors for each category

TL: Bar chart for errors of top 10 types

ZN: Category-level error co-occurrence

ZN: Accuracy by keywords (ORDER BY, GROUP BY, JOIN...)

ZN: Venn diagram on model overlapping

ZN: The similarity among errors made in the same NLQ by different model

ZN: Edit distance (Levenshtein distance)

- Top5 category for SmBop: B,L,C,E,F
- Top5 category for BRIDGE: B,L,D,C,F
- Top5 category for GAZP: B,L,C,E,F
- G,H,I,M only exists in SmBop
- Top5 type for SmBop: BL1, CL1, CE2, D1, F4
- Top5 type for BRIDGE: D1, D1a, CL1, BL1, F4
- Top5 type for GAZP: BL1, CL1, CE2, D1, A1b
-

The result shows that:.....

4 STUDY 2

TL: a *controlled* study to understand the effectiveness of different interaction strategies on different types of errors. a comprehensive design space that covers approaches proposed in prior work [examples]

4.1 Study Procedure

4.1.1 Participants.
TL: report basic demographics (gender, age, profession), participant expertise level with SQL

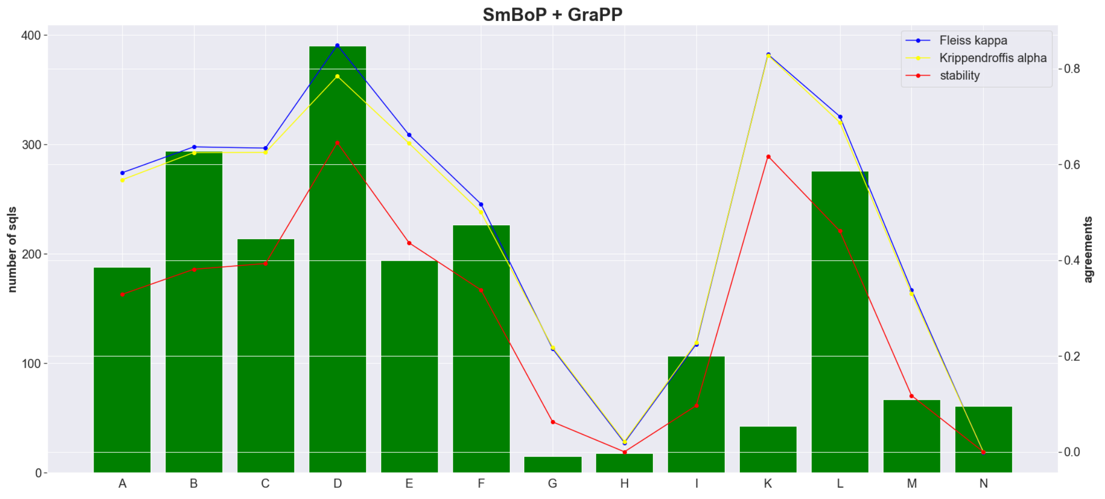

Fig. 1. This figure shows the number of times that each error category occurs in SmBoP + GraPP model (and the agreement score of each error category).

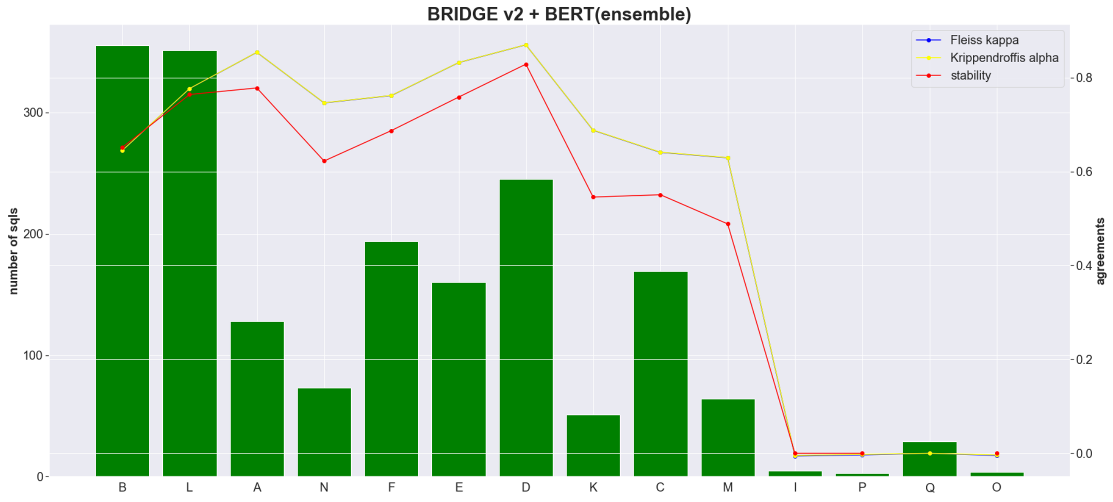

Fig. 2. This figure shows the number of times that each error category occurs in BRIDGE v2 + BERT(ensemble) model (and the agreement score of each error category).

We recruited 16 participants with 8 men and 8 women, aging from xx to xx from university mailing list. Each participant was compensated with $xx USD for their time. Each participant took part in the study that contains 4 conditions, one is the baseline condition and three others are experiment conditions. The experimental conditions and NL-SQL pairs are randomly chosen for each participant. The hardness distribution of the queries for each participant is the same. A complete study lasted 70 minutes. Each participant spent 15 minutes on annotating errors under each condition, and

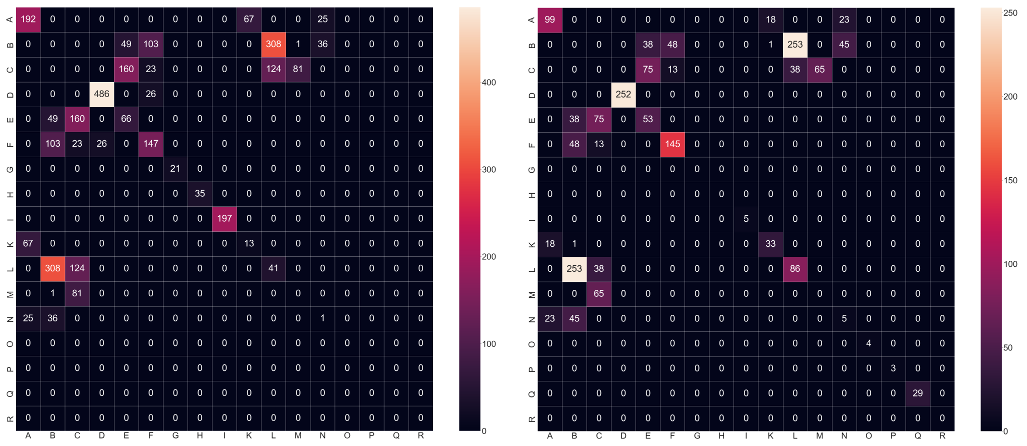

Fig. 3. This figure shows the number of times that each error type occurs in each of the three models: SmBoP + GraPP model, BRIDGEv2 + BERT(ensemble) model, and XX model.

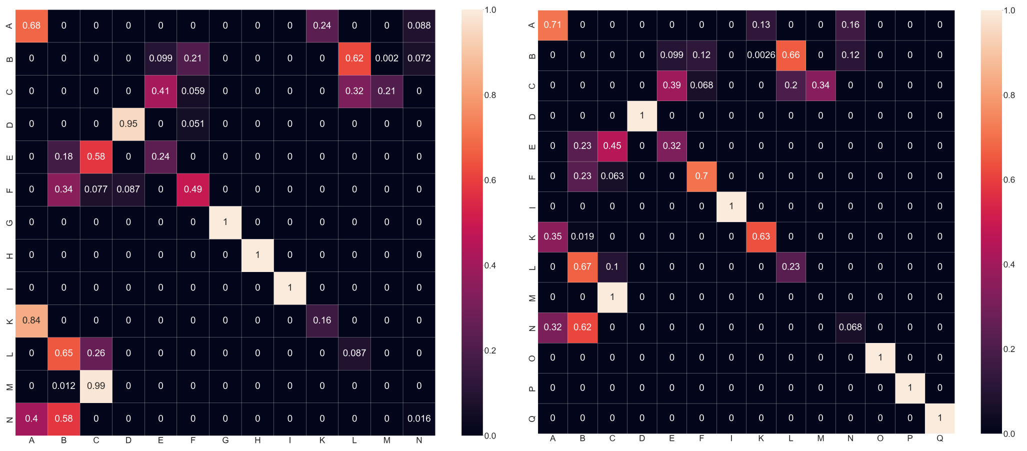

Fig. 4. This figure shows the probability of the number of times that error type occurs regarding the occurrences of each category for each of the three models: SmBoP + GraPP model, BRIDGEv2 + BERT(ensemble) model, and XX model.

ended with a 10-minute semi-structured interview where they reflected on their experience. All user study sessions were video recorded.

4.2 Conditions

TL: a table of interaction strategies/techniques included in the conditions

We used three strategies for error discovery: (1) SQLvis cite xxx, which generate a graph-based visualization of SQL queries; (2) NLExplain cite xxx , which executes the generated SQL statement step-by-step and provide natural language explanations for each step; (3) Sample table, where users can preview the schema and content of the selected tables. Two

	Error Discovery	Error Repair
Baseline Cond.	Baseline	DIYRepair
Baseline Cond. 2?	SampleTable	DIYRepair
Cond. #1	SQLViz + SampleTable	DIYRepair
Cond. #2	NLExplation + SampleTable	DIYRepair
Cond. #3	NLExplation	DIYRepair
Cond. #4	MISP	MISP + DIYRepair

strategies are applied for error repair: (1) Interactive direct manipulation of SQL queries; (2) Example-based interface where the system infer SQL queries from users and specify desired queries results using program synthesis.

A table of interaction strategies/techniques included in the conditions is showed in table xxx

4.3 Results

4.4 System Implementation

TL: technical details of how we replicated the interfaces in prior literature

5 DISCUSSION

5.1 Design Implications

6 LIMITATIONS AND FUTURE WORK

7 CONCLUSION

REFERENCES

[1] Axel Antoine, Sylvain Malacria, Nicolai Marquardt, and Géry Casiez. 2021. *Interaction Illustration Taxonomy: Classification of Styles and Techniques for Visually Representing Interaction Scenarios.* ACM, New York, NY, USA. https://doi.org/10.1145/3411764.3445586

[2] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2012. Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing.(2020), 4870–4888.

[3] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and Inter-Rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice. *Proc. ACM Hum.-Comput. Interact.* 3, CSCW, Article 72 (nov 2019), 23 pages. https://doi.org/10.1145/3359174

[4] Ohad Rubin and Jonathan Berant. 2020. SmBoP: Semi-autoregressive bottom-up semantic parsing. *arXiv preprint arXiv:2010.12412* (2020).

[5] Pengcheng Yin and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. *arXiv preprint arXiv:1810.02720* (2018).

[6] 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. 2018. 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.* Association for Computational Linguistics, Brussels, Belgium.

[7] 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. 2018. 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.* ACL, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425

[8] Victor Zhong, Mike Lewis, Sida I Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. *arXiv preprint arXiv:2009.07396* (2020).