An Empirical Study of Model Errors and User Error Discovery and Repair Strategies in Natural Language Database Queries

ZHENG NING[*], University of Notre Dame, USA
ZHENG ZHANG[*], University of Notre Dame, USA
TIANYI SUN, University of Chicago, USA
YUAN TIAN, Purdue University, USA
TIANYI ZHANG, Purdue University, USA
TOBY JIA-JUN LI, University of Notre Dame, USA

Recent advances in machine learning (ML) and natural language processing (NLP) have led to significant improvement in natural language interfaces for structured databases (NL2SQL). Despite the great strides, the overall accuracy of NL2SQL models is still far from being perfect (~75% on the Spider benchmark). In practice, this requires users to discern incorrect SQL queries generated by a model and manually fix them when using NL2SQL models. Currently, there is a lack of comprehensive understanding about the common errors in auto-generated SQLs and the effective strategies to recognize and fix such errors. To bridge the gap, we (1) performed an in-depth analysis of errors made by three state-of-the-art NL2SQL models; (2) distilled a taxonomy of NL2SQL model errors; and (3) conducted a within-subjects user study with 26 participants to investigate the effectiveness of three representative interactive mechanisms for error discovery and repair in NL2SQL. Findings from this paper shed light on the design of future error discovery and repair strategies for natural language data query interfaces.

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:
Zheng Ning, Zheng Zhang, Tianyi Sun, Yuan Tian, Tianyi Zhang, and Toby Jia-Jun Li. 2023. An Empirical Study of Model Errors and User Error Discovery and Repair Strategies in Natural Language Database Queries. In *28th International Conference on Intelligent User Interfaces (IUI '23), March 27-31, 2023, Sydney, Australia.* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3491102.3517479

1 INTRODUCTION

Data querying is an indispensable step in data analysis, sensemaking, and decision-making processes. However, traditional data query interfaces require users to specify their queries in a formal language such as SQL, leading to significant learning barriers for non-expert users who have little programming experience [46, 49]. This problem becomes increasingly important in the Big Data era, given the rising needs for end users in many key domains including business, healthcare, public policy, and scientific research. To address this problem, natural language (NL) data query interfaces allow users to express data queries in natural language. For example, a semantic parser can parse the user's

[*]These authors contributes equally to this work.

natural language query into a formal data query language such as SQL (NL2SQL). Such natural language interfaces have shown the potential to lower the bar for data querying and support flexible data exploration for end users [5, 54].

However, achieving robust NL2SQL parsing in realistic scenarios is difficult because of the ambiguity in natural language and the complex structures (e.g., nested queries, joined queries) in the target queries. For example, in *Spider* [63], a large-scale complex and cross-domain dataset for NL2SQL parsing, the accuracy of state-of-the-art models remained low in the 20% to 30% range for quite some time until 2019[1]. Recently, advances in deep learning (DL) have brought us closer than ever to achieving useful performance on this important task. With the use of large pre-trained models, the performance of state-of-the-art models [16, 23, 29] quickly increased to about 75% in the past two years. However, the development in model performance appears to have stagnated in the 75% range in the last year, suggesting a bottleneck in model-only methods for NL2SQL.

Furthermore, the flip side of an accuracy of 75% is an error rate of 25%. This high error rate hinders the adoption of NL2SQL models in the real world. When an error occurs, it is difficult for users to discover the error and repair it due to the "black-box" nature of DL models. Because data querying is often the foundation of a data work pipeline, any error in the query can misinform the following data analysis and decision-making process, causing catastrophic downstream impacts, especially in high-stake domains. Most prior practices safeguarding users from AI's failure are meant to integrate user-friendly interactive debugging mechanisms that allow users to easily detect and repair AI's breakdown through human-AI collaboration. For example, in the domain of task-oriented conversational agents, SOVITE [35] allows users to fix agent breakdowns in understanding user instructions by revealing the agent's current state of understanding of the user's intent and supporting direct manipulation for the user to repair the detected errors. However, for NL2SQL, a major gap in the literature is the lack of empirical studies on the types of errors that state-of-the-art NL2SQL models make, which blindfolds the design of effective error handling mechanisms for NL2SQL.

In the HCI and NLP communities, there are some recent efforts to support user understanding, error detection, and error repair of SQL queries during the NL2SQL process using different approaches. For example, following the *task decomposition* paradigm, DIY [40] decomposes a generated SQL statement into a sequence of incremental subqueries and provides step-by-step explanations to help users identify errors. Meanwhile, QueryVis [31], QUEST [8], and SQLVis [39] seek to improve user understanding of the generated SQL statements by visualizing the structures and relations among entities and tables in a user-friendly way. The conversational method is also used to communicate the current state of the model with users and update the results with user feedback through dialogs [21, 58, 61]. Although these methods were shown to be useful in different contexts in individual evaluations, it is not clear how effective each method is with respect to different error types, which hinders adaptive interaction design for NL2SQL debugging tools. Most of these methods were also evaluated with only simple NL2SQL errors, so it is unclear how well they will perform on errors made by state-of-the-art NL2SQL models on complex datasets such as Spider [63].

To bridge those gaps, we first conducted a comprehensive analysis of SQL errors made by state-of-the-art NL2SQL models and developed an axial taxonomy of those errors. In particular, we reproduced three representative models from the Yale Spider leaderboard — SmBop [43], BRIDGE [37], GAZP [67]. For each model, we collected all model-generated queries whose execution results are different from the ground truth queries. Four authors conducted multiple rounds of coding and refinement on these errors to derive a taxonomy of NL2SQL errors. The error analysis result shows that though the structure and performance of the models varied, these models made errors on a common set of queries and

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

showed a similar distribution among different error types. The findings can not only inform the design of future NL2SQL error-handling mechanisms, but also help machine learning practitioners identify opportunities to improve their models.

We also conducted a controlled user study ($N = 26$) to investigate the effectiveness and efficiency of representative error discovery and repair methods, including (i) a hybrid explanation- and example-based approach that supports fixing the SQL query through entity mapping between the natural language (NL) question and the generated query and discovering the error through a step-by-step NL explanation approach (DIY [40]), (ii) an explanation-based SQL visualization approach (SQLVis [39]), and (iii) a conversational dialog approach [61]. The study reveals that these error-handling mechanisms have limited impacts on increasing the efficiency and accuracy of error discovery and repair for errors made by state-of-the-art NL2SQL models. Finally, we discussed the implications for future error-handling mechanisms in natural language query interfaces.

To conclude, this paper presents the following three contributions:

- We developed a taxonomy of error types for three representative state-of-the-art NL2SQL models through iterative and axial coding procedures.
- We conducted a controlled user study that investigated the effectiveness and efficiency of three representative NL2SQL error discovery and repair methods.
- We discussed the implications for designing future error-handling mechanisms in natural language query interfaces.

2 RELATED WORK

2.1 NL2SQL techniques

Supporting natural language queries for relational databases is a long-standing problem in both the DB and NLP communities. Given a relational database D and a natural language query q_{nl} to D, an NL2SQL model aims to find an equivalent SQL statement q_{sql} to answer q_{nl}. The early methods of mapping q_{nl} to q_{sql} depend mainly on the development of intermediate logical representation [19, 59] or mapping rules [5, 32, 41, 44, 60]. In the former case, q_{nl} is first parsed into logical queries independent of the underlying database schema, which are then converted into queries that can be executed on the target database [25]. On the contrary, rule-based methods generally assumed that there is a one-to-one correspondence between the words in q_{nl} and a subset of database keywords/entities [25]. Therefore, the NL2SQL mapping can be achieved by directly applying the syntactic parsing and semantic entity mapping rules to q_{nl}. Although both strategies have achieved significant improvement over time, they have two intrinsic limitations. First, they require significant effort to create hand-crafted mapping rules for translation [25]. Second, the coverage of these methods is limited to a definite set of semantically tractable natural language queries [25, 41].

The recent development of deep learning (DL) based methods aims to achieve flexible NL2SQL translation through a data-driven approach [9, 20, 24, 37, 43, 67, 68]. From large-scale datasets, DL-based models learn to interpret NL queries conditioned on a relational DB via SQL logic [37]. Most NL2SQL models use the encoder-decoder architecture [37, 67, 68], where the encoder models the input q_{nl} into a sequence of hidden representations along time steps. The decoder then maps the hidden representations into the corresponding SQL statement. Recently, Transformer-based architecture [37, 57] and pre-training techniques [22, 47, 62] have become popular as the backbone of NL2SQL encoders. At the same time, many decoders have been used to optimize SQL generation, such as autoregressive bottom-up decoding [43] and the LSTM-based pointer-generator [37]. However, those DL-based models are usually "black-boxes" due to the lack of explainability [25]. The lack of transparency makes it difficult for users to figure out how to fix the observed errors when using DL-based NL2SQL models.

The evaluation of these DL-based models is based mainly on objective benchmarks such as Spider [63] and WikiSQL [68]. For example, Spider requires models to generalize well not only to unseen NL queries but also to new database schemas, in order to encourage NL interfaces to adapt to cross-domain databases. The performance of a model is evaluated using multiple measures, including component matching, exact matching, and execution accuracy. However, these benchmarks only involve quantitative analysis of NL2SQL models, giving little clue about what types of errors a model tends to fall into.

An aim of our work is to develop a taxonomy of error types of errors made by state-of-the-art NL2SQL models and report the corresponding descriptive statistics to complement the quantitative benchmark with the qualitative analysis of those NL2SQL models.

2.2 Detecting and repairing errors for NL2SQL

Natural language interfaces for NL2SQL face challenges in language ambiguity, underspecification, and model misunderstanding [13, 40]. Previous work has explored ways to support error detection and repair for NL2SQL systems through human-AI collaboration. NL2SQL error detection methods can be mainly divided into categories of natural language-explanation-based, visualization-based, and conversation-based approaches. NL2SQL error repair methods consist mainly of direct manipulation and conversational error fixing approaches.

For error detection, a popular method is to explain the query and its answer in natural language [13, 50, 52, 61]. For example, DIY [40] shows step-by-step NL explanations and query results by applying templates to subqueries, helping users understand SQL output in an incremental way; NaLIR [33] explains the mapping relations between entities in the input query and those in the database schema; Ioannidis et al. [50] introduced a technique to describe structured database content and SQL translation textually to support user sensemaking of the model output. Visualizations have also been widely used to explain a SQL query and its execution [7, 8, 31, 39]. For example, QueryVis [31] produces diagrams of SQL queries to capture their logical structures; QUEST [8] visualizes the connection between the matching entities from the input query and their correspondences in the database schema; SQLVis [39] introduced visual query representations to help SQL users understand the complex structure of SQL queries and verify their correctness.

Most of previous work employed direct manipulation to repair and disambiguate queries. NaLIR [33], DIY [40] and DataTone [17] allow users to modify the entity mappings through drop-downs; Eviza [48] and Orko [51] enable users to modify quantitative values in queries through range sliders. In addition to direct manipulation, several other prior interaction mechanisms enable users to give feedback to NL2SQL models through dialogs in natural language. For example, MISP [61] maintains the state of the current parsing process and asks for human feedback to improve SQL translation through human-AI conversations. Elgohary et al. [13, 14] investigated how to enable users to correct NL2SQL parsing results with natural language feedback in conversation.

With the many error-handling mechanisms that have been proposed, there is a gap in evaluating how effective and efficient these mechanisms are to address different types of NL2SQL errors and what specific limitations they have. These types of information are critical to inform the effective choice and design of NL2SQL error handling mechanisms in different use scenarios and to inspire the use of ensemble mechanisms to handle different usage contexts of NL2SQL. Our work bridges this gap by investigating these questions through controlled user studies, whose findings could guide the future design of NL2SQL error handling systems.

2.3 Error handling via human-AI collaboration

Handling errors made by AI models in human-AI collaboration faces many key challenges. First, many state-of-the-art AI models lack transparency in their decision-making process, making it difficult for users to understand exactly what leads to incorrect predictions [45]. Although there are some attempts to explain the state of the AI model using methods such as heatmap [42, 69], search traces [64], and natural language explanations [11, 12], they only allow users to peek at the AI model's reasoning at certain stages instead of exposing the holistic states of the model. Second, it is difficult for users to develop a correct mental model for complex AI models due to the *representational mismatch* in which "*humans can create a mental model in terms of features that are not identical to those used by AI models*" [6]. Lastly, error handling usually requires multiple turns of interactions [27, 35]. However, maintaining coherent multi-turn interactions between AI and humans is challenging [1]. It requires AI to closely maintain and update the context history, evolve its contextual understanding, and behave appropriately based on user's timely responses [2, 3, 70].

Our work contributes to the knowledge of how users handle errors in their collaborations with NL2SQL models by studying how users utilize existing error-handling mechanisms to inspect and fix errors made by NL2SQL models and how they perceive the usefulness of these mechanisms. Our findings of user challenges also echo the identified challenges in human-AI collaborations in other domains (e.g., programming [55, 56, 65], data annotation [18], QA generation [66], interactive task learning [34, 36]), showing that users need help comprehending the state of AI models and developing a proper mental model in AI-based interactive data-centric tools to understand and assess their recommendations.

3 AN ANALYSIS AND TAXONOMY OF NL2SQL ERRORS

In this section, we describe the development of the taxonomy of NL2SQL errors of three representative state-of-the-art NL2SQL models and the corresponding error analysis. The structure of this section is as follows. Section 3.1 summarizes the methodology used for building the dataset; Section 3.2 explains the axial and iterative coding procedure we used to derive the error taxonomy; Section 3.3 describes the developed taxonomy of NL2SQL errors; Section 3.4 presents an analysis of erroneous queries in the dataset based on the taxonomy.

3.1 Dataset collection

We adopted the Spider [63] dataset to train and evaluate the models to collect a set of erroneous SQL queries for the taxonomy. Spider is the most popular benchmark to evaluate NL2SQL models with complex and cross-domain semantic parsing problems. The original Spider dataset consists of around 10,000 queries in natural language on multiple databases across different domains (e.g., "soccer", "college"). Depending on the complexity of their structures and the SQL keywords involved, the queries are divided into four difficulty levels: "Easy", "Medium", "Hard", and "Extra Hard". In this work, we chose to focus only on the first three difficulty levels, since state-of-the-art models have significantly lower accuracy on the "extra hard" queries. For example, the best model we reproduced, SmBop [43], only achieved ~50% in accuracy), indicating that NL2SQL for "extra hard" queries remains less feasible at this point.

Since the held-out test set in Spider is not publicly available, we created our own test set by re-splitting the public training and development sets from Spider. The ratios of the three difficulty levels in the new training and testing sets were similar to those in the original training and developing sets, respectively. In addition, we ensured that there is no overlap in the databases used between the training and testing sets. Table 1 shows the distribution of our training and test data compared to the original public Spider dataset. We then re-trained three NL2SQL models from the official

		Easy	Medium	Hard	Extra	Total
Original	Train	1983	2999	1921	1755	8658
	Dev	248	446	174	166	1034
Re-split	Train	1604	2363	1516	0	5483
	Test	627	1082	579	0	2288

Table 1. Descriptive statistics of the original Spider dataset and our sampled dataset

Models	Err. queries	Retrained Acc.	Original Acc.
SmBop	431	81.2%	71.1%
BRIDGE	853	62.7%	68.3%
GAZP	1062	53.6%	53.5%

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

Spider leaderboard. We chose models that officially released the code, used different core structures, and showed close to SOTA performance at the time we prepared the dataset. These models are SmBoP [43], BRIDGE [37] and GAZP [67].

The erroneous queries are those that have different execution results from the correct ones. The queries generated by these models on the test set were manually analyzed to develop the taxonomy of SQL generation errors. Table 2 shows the total number of erroneous queries generated by each model. The accuracy of each model on our test set is similar to the reported performance of these models on the private held-out test set, indicating that our reproduction and retraining of these models are consistent with the originals[2].

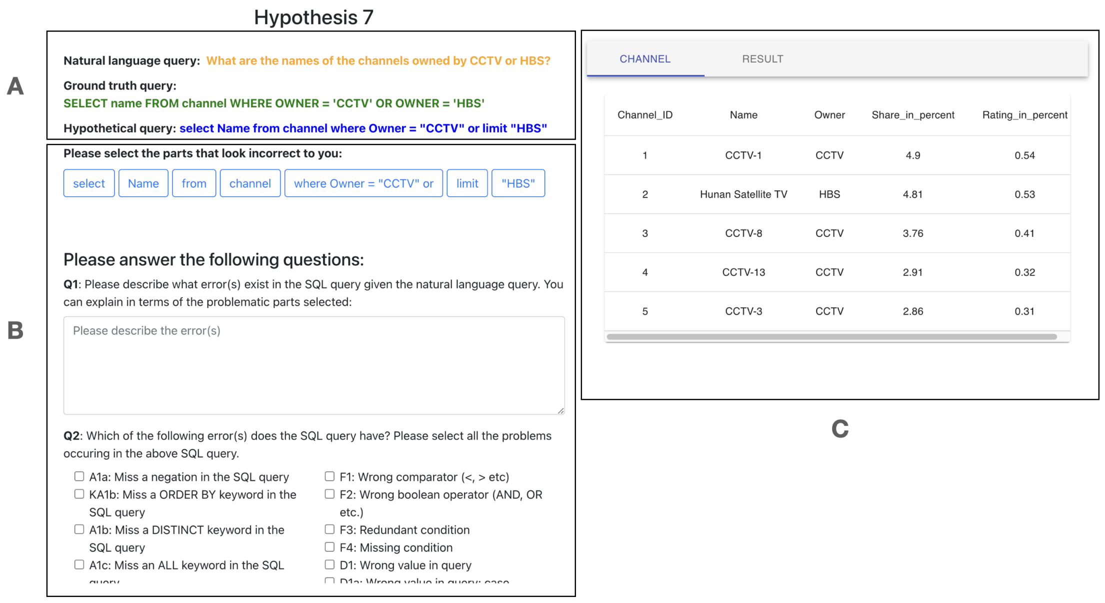
3.2 The coding procedure

After curating the dataset of NL2SQL errors, we followed the established open, axial and iterative coding process [4, 28] to develop a taxonomy of NL2SQL errors. The detail of the process is as follows.

3.2.1 Step 1: Open coding. To begin with, we randomly sampled 40 erroneous SQL queries to develop the preliminary taxonomy. Four authors with in-depth SQL knowledge performed open coding [10, 28] on this subset of erroneous SQL queries. They were instructed to code to answer the following questions: (1) *What are the errors in the generated SQL query in comparison to the ground truth?* (2) *What SQL component does each error reside at?* (3) *Have all the errors in the incorrect SQL query been covered?*. Once we finished the first round of coding, we put all the coded query pairs (the generated query and the ground truth) line by line in a shared spreadsheet. The annotators sat together to discuss the codes and reached a consensus in the preliminary version of the codebook.

3.2.2 Step 2: Iterative refinement of the codebook. After creating the preliminary codebook, four annotators conducted iterative refinements of the established codes. Each iteration consisted of the following three steps. First, the annotators coded a new sample batch of 40 unlabeled erroneous queries using the codebook from the last iteration. If there is a new error not covered by the current codebook, annotators would write a short description of it. Second, we computed the inter-rater reliability between coders [38] (Fleiss' Kappa and Krippendorff's Alpha) at the end of each iteration. Lastly, annotators exchanged opinions about adding, merging, removing, or reorganizing codes and updated the codebook accordingly. Annotators completed three refinement iterations until the codebook became stable and the inter-rater

[2]The data, the replicated models, and the evaluation scripts we used are published in https://github.com/realningzheng/IUI23-NLQ



Natural language query: What are the names of the channels owned by CCTV or HBS?

Ground truth query:
SELECT name FROM channel WHERE OWNER = 'CCTV' OR OWNER = 'HBS'

Hypothetical query: select Name from channel where Owner = "CCTV" or limit "HBS"

Please select the parts that look incorrect to you:

Please answer the following questions:

Q1: Please describe what error(s) exist in the SQL query given the natural language query. You can explain in terms of the problematic parts selected:

Please describe the error(s)

Q2: Which of the following error(s) does the SQL query have? Please select all the problems occuring in the above SQL query.

- ☐ A1a: Miss a negation in the SQL query
- ☐ KA1b: Miss a ORDER BY keyword in the SQL query
- ☐ A1b: Miss a DISTINCT keyword in the SQL query
- ☐ A1c: Miss an ALL keyword in the SQL query
- ☐ F1: Wrong comparator (<, > etc)
- ☐ F2: Wrong boolean operator (AND, OR etc.)
- ☐ F3: Redundant condition
- ☐ F4: Missing condition
- ☐ D1: Wrong value in query
- ☐ D1a: Wrong value in query; case

Channel_ID	Name	Owner	Share_in_percent	Rating_in_percent
1	CCTV-1	CCTV	4.9	0.54
2	Hunan Satellite TV	HBS	4.81	0.53
3	CCTV-8	CCTV	3.76	0.41
4	CCTV-13	CCTV	2.91	0.32
5	CCTV-3	CCTV	2.86	0.31

Fig. 1. The user interface that we used for NL2SQL error annotation

reliability scores were substantial. At the end of the final refinement iteration, the Fleiss' Kappa was 0.69 and the Krippendorff's Alpha was 0.67.

3.2.3 Step 3: Coding the remaining dataset. We then proceeded to code the remaining dataset using the codebook from the final refinement iteration. Because the inter-rater reliability scores stabilized among annotators, two annotators participated in this step. The Fleiss' Kappa and Krippendorff's Alpha of the full dataset annotation between those two annotators were 0.76 and 0.78 respectively, indicating substantial agreement [4, 15, 26].

3.2.4 The annotation interface. The interface used for the annotation of the NL2SQL errors is shown in Figure 1. It consists of three components: (1) a natural language query, the corresponding ground truth, and the model-generated SQL query are displayed in A; (2) In the error annotation section, the annotator first decided which part(s) of the generated SQL is wrong in B, after that, the annotator was supposed to choose error types from the checkbox below. If the error type was not included, the system provided an input box to provide open feedback. We updated the error types after each batch of coding. (3) The tables involved in the pairs and the query result table were displayed on the right side of the canvas to help annotators identify the error types.

3.3 The Taxonomy of NL2SQL Errors

Table 3 shows the finalized taxonomy of NL2SQL errors. Specifically, we categorized the error types along two dimensions: (1) the *syntactic* dimension shows which parts of the SQL query an error occurs in, categorized by SQL keywords such as `WHERE` and `JOIN`; (2) the *semantic* dimension indicates which aspects of the NL description that the model misunderstands, such as misunderstanding a value or the name of a table. For each type of error, the uppercase letter refers to the syntactic category, and the lowercase letter refers to the semantic category. Note that there may be multiple manifestations of a semantic error in a syntactic error category. For example, the table error has two different forms in the "JOIN" clause, including "`Missing a table to JOIN`" (Ba1) and "`JOIN the wrong table`" (Ba2). An erroneous query may also have multiple error types associated with it.

Error categories	Error types			SmBop	BRIDGE	GAZP
Syntactic errors	A: WHERE error	Aa1:	Use a wrong table in WHERE	21	68	73
		Ab1:	Use a wrong column in WHERE	19	36	23
		Ac1:	Redundant WHERE clause	14	16	27
		Ac2:	Missing WHERE clause	15	21	61
		Ad1:	Other wrong value in WHERE clause	51	52	91
		Ad2:	Value case error in WHERE clause	62	69	82
		Ad3:	Value plurality error in WHERE clause	8	6	16
		Ad4:	Value synonym error in WHERE clause	35	40	45
		Ae1:	Wrong comparator (<, >, =, !=, etc)	8	13	14
		Ae2:	Wrong boolean operator (AND, OR etc.)	4	15	9
	B: JOIN error	Ba1:	Miss a table to JOIN	35	106	101
		Ba2:	JOIN the wrong table	24	89	78
		Bb1:	Use a wrong column in JOIN	13	79	69
		Bc1:	Redudant JOIN clause	17	82	113
	C: ORDER BY error	Cb1:	Use a wrong column to sort	3	26	26
		Cc1:	Miss a ORDER BY clause	12	22	20
		Cc2:	Redundant sorting	3	1	3
		Ce1:	Wrong sorting direction	6	27	23
	D: SELECT error	Da1:	Use a wrong table in SELECT	59	117	106
		Db1:	Return a wrong column in SELECT	21	56	78
		Db2:	Return a redundant column in SELECT	10	19	36
		Db3:	Miss returning column(s) in SELECT	20	34	59
		Df1:	Use wrong aggretation function	7	43	11
		Df2:	Miss aggregation function	5	19	14
	E: GROUP BY error	Eb1:	Use a wrong column in GROUP BY	6	10	18
		Ec1:	Miss a GROUP BY clause in the SQL query	10	33	47
		Ec2:	Redudant GROUP BY clause	6	7	16
	F: HAVING error	Fc1:	Miss HAVING clause	1	5	12
		Fc2:	Redundant HAVING clause	0	2	6
		Fe1:	Wrong condition in HAVING	1	2	3
	G: LIKE error	Gc1:	Miss LIKE clause	1	3	9
		Ge1:	Wrong LIKE condition	1	8	22
	H: LIMIT error	Hc1:	Redudant LIMIT clause	1	2	6
		Hc2:	Miss LIMIT clause	0	1	3
	I: INTERSECT error	Ie1:	Wrong INTERSECT condition	8	8	9
	J: DISTINCT error	Jc1	Miss a DISTINCT keyword	7	18	96
		Jc2	Redundant DISTINCT keyword	4	15	0
	K: EXCEPT error	Kc1	Wrong EXCEPT clause	14	27	24
	L: NOT error	Lc1	Miss NOT keyword	7	9	7
	M: UNION	Me1	Wrong UNION condition	9	9	8
Semantic errors	a: Table error			97	279	274
	b: Column error			81	237	251
	c: Miss/redundant Clause/keyword error			107	250	432
	d: Value error			153	162	230
	e: Condition error			37	82	88
	f: Aggregation function error			12	62	25

Table 3. The taxonomy of NL2SQL errors with the count of errors of each type for the three models

3.4 NL2SQL error analysis

Based on the SQL generation error taxonomy, we conducted an analysis on the set of erroneous queries to investigate the following three questions.

- **Q1**: How are the erroneous queries distributed among different models? Do models tend to stumble on the same queries or make mistakes on distinct queries? For those overlapping erroneous queries, do models tend to make similar types of error on them or not?
- **Q2**: How do error types spread along the syntactic and semantic dimensions? How different are the distributions of error types among the three models?
- **Q3**: How far are the erroneous queries from their corresponding ground truths?

3.4.1 The distribution of erroneous queries among models. Figure 2 shows the overlap of erroneous queries among the three models in a Venn diagram, where each circle represents the queries on which the model made errors. The size of each circle is proportional to the number of erroneous queries of its corresponding model in the sampled dataset (Table 2). Table 4 shows the number of queries in each intersection and union area. In addition, 81.3% (350 out of 431) of SmBop's incorrect queries and 83.5% (712 out of 853) of BRIDGE's incorrect queries also confounded other models. The results imply that ***different models tend to make errors on the same subset of queries in NL2SQL.***

$$Jaccard\ distance = \frac{|E_{smbop} \cap E_{bridge} \cap E_{gazp}|}{|E_{smbop} \cup E_{bridge} \cup E_{gazp}|} \tag{1}$$

To understand whether models make similar types of errors in those overlapped queries, for each query in which all models made errors, we calculated the Jaccard distance [30] of the syntactic and semantic error types that different models made on this query. Jaccard distance measures how similar multiple sets are. The definition of Jaccard distance is shown in Equation 1, where E_m means the set of error types that the model m made on the target query. For syntactic error types, 21.7% of overlapped queries have a Jaccard distance of 0 among the three models, which implies that the models did not all make the same syntactic error type in these queries. Regarding the semantic error types, 35.5% of these queries have a Jaccard distance of 0. On the other hand, only 8.7% of the overlapped queries have the same syntactic error type from the three models, and even fewer of them (4.3%) have the same semantic error type from all three models. These results show that ***although the models tend to make errors in the same set of queries, the types of errors in each query tend to be different***. We provide three examples in Table 5 to illustrate the disparity of error types in the same queries.

	Count Intersect	**Count Union**
SmBop-BRIDGE	307	977
SmBop-GAZP	319	1174
BRIDGE-GAZP	681	1234
SmBop-BRIDGE-GAZP	276	1315

Table 4. The numbers of erroneous queries corresponding to Figure 2

3.4.2 Error frequency. In this section, we investigate the distribution of error types among models. Specifically, we report the following three measures for each model:

(1) **Syntactic error rate ($SYNER_{ms}$)**: Given the model m and a syntactic error type s, $SYNER_{ms}$ is the number of queries in which the model m made the syntactic error s divided by the number of ground truth queries in the

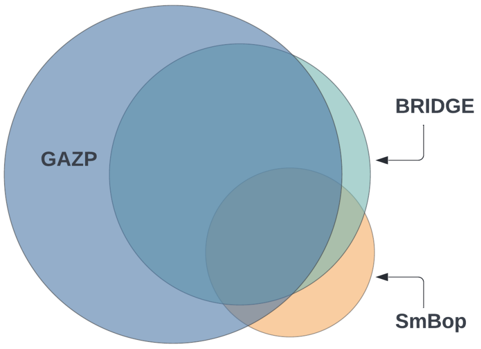


Fig. 2. The overlap of erroneous queries generated by three NL2SQL models

	1		2		3	
Natural language query	Show all product sizes.		What are the names of the states that have 2 to 4 employees living there?		What are the names of all video games that are collectible cards?	
Ground truth query	SELECT DISTINCT product_size FROM Products		SELECT T1.state_province_county FROM Addresses AS T1 JOIN Staff AS T2 ON T1.address_id = T2.staff_address_id GROUP BY T1.state_province_county HAVING count(*) BETWEEN 2 AND 4;		SELECT gname FROM Video_games WHERE gtype = "Collectible card game"	
Model	**Generated query**	**Error types**	**Generated query**	**Error types**	**Generated query**	**Error types**
SmBop	SELECT DISTINCT products.product_color FROM products	Db1	SELECT addresses.state_province_county FROM addresses JOIN staff ON addresses.address_id = staff.staff_address_id GROUP BY addresses.state_province_county HAVING COUNT(*) >= 2	Fe1	SELECT video_games.gname FROM video_games WHERE video_games.gtype = 'collectible cards'	Ad1, Ad4
BRIDGE	SELECT Products.product_size FROM Products	Jc1	SELECT Addresses.state_province_county FROM Addresses GROUP BY Addresses.state_province_county HAVING COUNT(*) >= 2	Fe1, Ba1	SELECT Video_Games.GName FROM Video_Games	Ac4
GAZP	select product_size from Products	Jc1	select count (*) group by having (*) between 4 and 2	Db1, Df1, Eb1, Fe1	select GName from Video_Games where GType = "videoible"	Ad1

Table 5. Sampled erroneous NL-SQL pairs and their error types

entire development set that has the corresponding syntax. (Table 6). It tells us how likely a syntactical part of a query will produce errors.

(2) **Syntactic error percentage (distribution) ($SYNEP_{ms}$)**: Given the model m and a syntactic error type s, $SYNEP_{ms}$ is the number of queries in which the model m made the syntactic error s divided by the total number of erroneous queries made by the model m. (Table 6). It measures the percentage of queries that contain a specific type of syntactic error among all erroneous queries.

(3) **Semantic error percentage (distribution) ($SEMEP_{ms}$)**: Given the model m and the semantic error rate s, $SEMEP_{ms}$ is the number of queries in which the model m made the semantic error s divided by the total number

	Error Percentage			Error Rate		
Error type	SmBop	BRIDGE	GAZP	SmBop	BRIDGE	GAZP
A: WHERE error	47.80%	35.05%	30.89%	18.86%	27.38%	30.04%
B: JOIN error	17.87%	28.14%	31.83%	10.13%	31.58%	44.47%
C: ORDER BY error	4.64%	7.39%	5.74%	4.58%	14.42%	13.96%
D: SELECT error	23.20%	25.79%	25.80%	4.37%	9.62%	11.98%
E: GROUP BY error	5.10%	5.86%	7.63%	4.50%	10.22%	16.56%
F: HAVING error	0.46%	1.06%	1.98%	1.44%	6.47%	15.11%
G: LIKE error	0.46%	1.29%	2.92%	2.86%	15.71%	44.29%
H: LIMIT error	0.23%	0.35%	0.85%	0.41%	1.24%	3.73%
I: INTERSECT error	1.86%	0.94%	0.85%	18.60%	18.60%	20.93%
J: DISTINCT error	2.55%	3.87%	9.04%	4.78%	14.35%	41.74%
K: EXCEPT error	3.25%	3.17%	2.26%	20.29%	39.13%	34.78%
L: NOT error	1.62%	1.06%	0.66%	13.46%	17.31%	13.46%
M: UNION error	2.09%	1.06%	0.75%	56.25%	56.25%	50.00%

Table 6. The error percentage and error rate of each syntactic error type

Error type	SmBop	BRIDGE	GAZP
a: Table error	22.51%	32.71%	25.80%
b: Column error	18.79%	27.78%	23.63%
c: Miss/redundant Clause/keyword error	24.83%	29.31%	40.68%
d: Value error	35.50%	18.99%	21.66%
e: Condition error	8.58%	9.61%	8.29%
f: Aggregation function error	2.78%	7.27%	2.35%

Table 7. The error percentage of each semantic error type

of erroneous queries made by the model *m*. (Table 7). It measures the percentage of queries that contain a specific type of semantic error among all erroneous queries.

As shown in Table 6, the distributions of syntactic error type are similar among all three models. Note that a model can produce a query with multiple types of errors. Notably, the error percentage of `WHERE`, `JOIN` and `SELECT` are significantly higher than that of other syntactic error types for all the models. However, comparing it with the syntactic error rate, we see that a higher frequency of errors (in all queries) does not equate to a higher error rate when a specific type of keyword is encountered. For example, although `UNION` errors only account for fewer than 3% of erroneous queries among all models, it has an error rate of more than 50% (i.e., when the correct query shoud contain a `UNION` clause, the model has a high probability of making errors there). The top-5 syntactic parts that have the highest error rates are shown in Table 8.

Compared to syntactic errors, the distribution of semantic errors is more varied between models (Figure 7). We found that `d: Value error`, `a: Table error` and `c: Miss/redundant clause/keyword error` are the most frequent error for SmBop (35.5%), BRIDGE (32.71%) and GAZP (40.68%) respectively. It indicates that ***the semantic challenges of the investigated NL2SQL models are more varied than the syntactic challenges they faced.***

3.4.3 Distance between erroneous and ground truth queries. Lastly, we used Levenshtein distance to measure the distance between erroneous and ground truth queries. The Levenshtein distance between two queries is defined as the

Model	Top 1	Top 2	Top 3	Top 4	Top 5
SmBop	UNION 56.25%	EXCEPT 20.29%	WHERE 18.86%	INTERSECT 18.60%	NOT 13.46%
BRIDGE	UNION 56.25%	EXCEPT 39.13%	JOIN 31.58%	WHERE 27.38%	INTERSECT 18.6%
GAZP	UNION 50.00%	JOIN 44.47%	LIKE 44.29%	DISTINCT 41.74%	EXCEPT 34.78%

Table 8. The top 5 error-prone syntactic parts of a SQL query

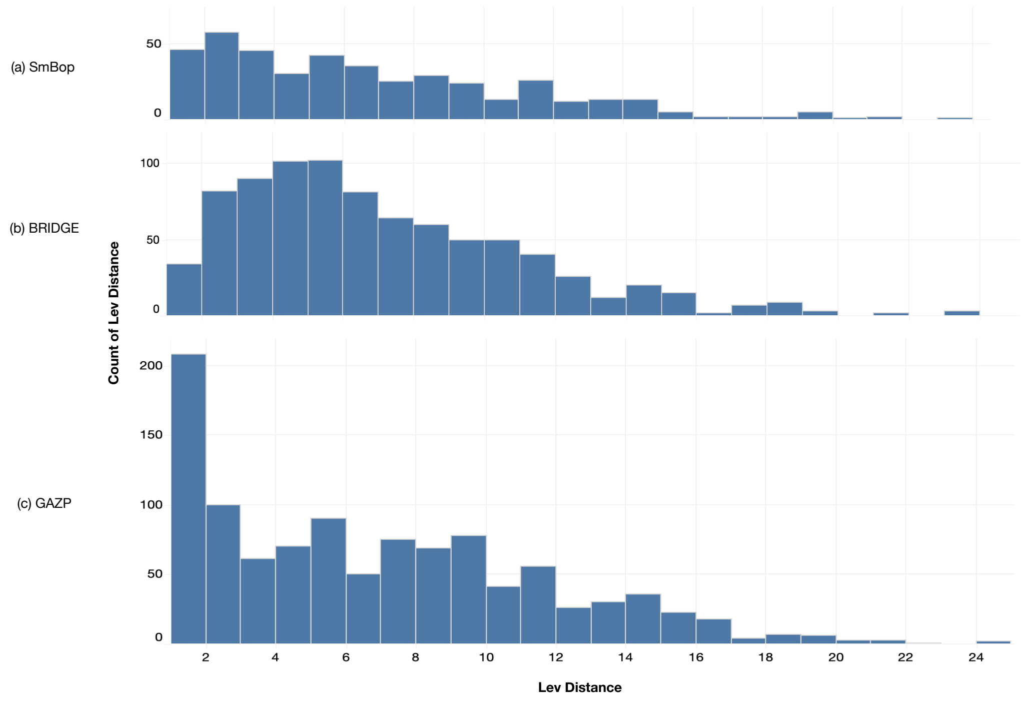


Fig. 3. The distribution of Levenshtein distances between erroneous queries and ground truth queries among three models

minimum number of word-level (split by space) edits (insertions, deletions, or substitutions) required to transform the model-generated query into the ground truth query.

Figure 3 shows the distribution of the Levenshtein distances of errors made by each model. It is worth noting that all three distributions have a long tail. Specifically, by looking at the Levenshtein distance in three different groups: 0–5, 6—10, and more than 10; we found that a large portion of erroneous SQL queries for all three models can be fixed in a small number of edits. In particular, 19.6% (208/1062) queries in GAZP only need changes in one token; the percentage of erroneous queries requiring only changes in one token for BRIDGE and SmBop is 3.9% (34/853) and 10.7% (46/431), respectively.

4 THE USER STUDY OF INTERACTIVE ERROR DISCOVERY & REPAIR MECHANISMS

In the past few years, we have seen a growing interest in interactive mechanisms for users to detect and repair NL2SQL errors [8, 21, 31, 39, 40, 58, 61]. To understand the performance and usage of these mechanisms by users, we conducted

Condition	Error Discovery and Repair Mechanisms
Baseline	Direct SQL query editing
Exp. Cond. #1	Step-by-step SQL query explanation & NL-SQL entity mapping (DIY [40])
Exp. Cond. #2	Graph-based SQL query visualization (SQLViz [39])
Exp. Cond. #3	Conversational dialog system (MISP [61])

Table 9. The list of conditions used in the user study

a controlled user study to examine the effectiveness of different error discovery and repair mechanisms for NL2SQL[3]. Specifically, we investigated the following research questions.

RQ1. How effective and efficient are the different error-handling mechanisms and interaction strategies in NL2SQL?
RQ2. What are the user preferences and perceptions of different mechanisms and strategies?
RQ3. What are the gaps between the capabilities of existing approaches and user needs?

4.1 Experiment conditions

In this study, we used four conditions shown in Table 9. In the baseline condition, no interactive support was provided for error discovery and repair. Users had to examine the correctness of a generated SQL query by directly checking the query result and manually editing a generated SQL query to fix an error.

In addition to the baseline, we selected three experimental conditions based on three representative approaches for error discovery and repair. The first experimental condition exemplifies an **explanation- and example-based** approach (DIY [40]) that displays intermediate results by decomposing a long SQL query into shorter queries and generating natural language explanations for each step. Meanwhile, it allows users to fix the mapping between words in the NL description and their corresponding entities in the generated SQL query from a drop-down menu. The second experimental condition uses an **explanation-based visualization** approach (SQLVis [39]). The technique uses a graph-based visualization for the generated SQL query to illustrate the explicit and implicit relationship among different SQL components such as the selected columns, tables, primary and foreign keys. The third experimental condition exemplifies a **conversational dialog approach** (MISP [61]). It allows users to correct an erroneous SQL query through multiple rounds of conversation in natural language (Table 9). We replicated the core functionalities of the DIY mechanism used in experimental condition #1 as described in the paper [40] because the official source code was not publicly released. For experimental condition #2, we used the official implementation[4] provided by the authors. For the dialog system under experimental condition #3, we implemented an interactive widget based on the open-sourced command-line tool and an interactive graphical user interface based on the React-Chatbot-Kit[5] for the study.

4.2 Participants

We recruited 26 participants from the campus community of a private university in the midwest of the United States through mailing lists and social media. Participants included 15 men and 11 women aged 20 to 30 years. Nine participants were novice SQL users who had either no experience in using SQL or had seen SQL queries before but were not familiar with the syntax. 10 participants were intermediate SQL users who had either taken an introductory database course

[3]The protocol of the study has been reviewed and approved by the IRB at our institution.
[4]https://github.com/Giraphne/sqlvis
[5]https://www.npmjs.com/package/react-chatbot-kit

or understood the SQL syntax. The remaining 7 were experienced users who were familiar with SQL queries or had significant experience working with SQL. Each participant was compensated with $15 USD for their time.

4.3 Study procedure

In our study, each participant experienced the four conditions described in Section 4.1. As the goal of this study is to investigate the error discovery and repair behavior of users, the example SQL queries for each participant were randomly selected from the dataset of incorrect queries generated by the three NL2SQL models used in the error analysis study. Each query that a participant encountered was also randomly assigned to one of the experimental conditions or the baseline condition.

To facilitate the user experiment, we implemented a web application that can automatically select SQL tasks and assign conditions to study participants. After finishing one SQL query, users can click the "Next" button on the application, and it will randomly select the next query and assign a condition to it. Both the query assignment and the condition assignment were randomized. For each query, the web application renders the task description, the database and its tables, and the assigned error-handling mechanisms.

Each experiment session began with the informed consent process. Then, each participant watched a tutorial video about how to interact with the system to solve an SQL task and fix NL2SQL errors under different conditions. Then, each participant was given a total of 45 minutes to solve as many SQL tasks as possible. On average, each participant completed 22.0 SQL tasks in 45 minutes (5.5 in each condition). After each experiment session, the participant completed a post-study questionnaire. This questionnaire asked participants to rate their overall experience, the usefulness of interactive tool support under different conditions, and their preferences in Likert scale questions. We ended each experiment session with a 10-minute semi-structured interview. In the interview, we asked follow-up questions about their responses to the post-study questionnaire, if they encountered any difficulties with interaction mechanisms under the conditions, and which parts they found useful. We also asked participants about the general workflow as they approached the task and the features they wished they had when handling NL2SQL errors. All user study sessions were video recorded with the consent of the participants.

Following established open coding methods [10, 28], an author conducted a thematic analysis of the interview transcripts to identify common themes about user experiences and challenges they encountered while using the different error handling mechanisms, as well as their suggestions for new features. Specifically, the coder went through and coded the transcripts of the interview sessions using an inductive approach. For user quotes that did not include straightforward key terms, the coder assigned researcher-denoted concepts as the code.

4.4 Data collection

For each SQL task, we collected three types of data from the participant: (1) the updated SQL query after their repair; (2) the starting and ending time; (3) the user's interaction log with the error handling mechanism (e.g., clicking to view the sampled table, opening the drop-down menu, interacting with the chatbot).

We cleaned up the data from the participants through the following steps. First, we filtered out the queries that are skipped by the participants (i.e., the user clicking on "Next" without making any changes to the query), which consist of less than 10% of the total data. Second, if the participant did not utilize the interaction mechanism associated with the experimental condition at all (e.g., the user inspected the query without using any assistance and modified the query manually), the task was deemed to be solved using the baseline method.

Conditions	Avg. Acc. (μ = 0.56)	SD (μ = 0.50)	Avg. ToC (μ = 116.7)	SD (μ = 89.6)
B1	0.55	0.48	109.7	95.8
C1	0.56	0.51	110.9	101.0
C2	0.60	0.50	115.9	96.5
C3	0.53	0.51	128.5	61.7

Table 10. The average accuracy and ToC (in seconds) for different conditions

4.5 Results

In this section, we report the key findings on the efficiency, effectiveness, and usability of different error handling mechanisms and their user experiences. For each condition in a statistical test, the data is sampled evenly and randomly.

F1: The error handling mechanisms do not significantly improve the accuracy of fixing erroneous SQL queries. To start with, we conducted a one-way ANOVA test (α=0.05) among tasks that used different error handling mechanisms. The p-value for the accuracy was 0.82, indicating that there were no significant differences between the different error handling methods. The average accuracy and standard deviation among the participants are shown in Table 10.

We then analyzed the effect of different mechanisms on the accuracy of fixing specific error types, including five common syntactic error types (`A: WHERE error; B: JOIN error; C: ORDER BY error; D: SELECT error; E: GROUP BY error`, as well as six semantic errors shown in Table 6. Using the same statistical test, we found that the p-values for all types of error were higher than the 0.05 threshold, indicating that there were no significant differences in accuracy when the user used different error-handling mechanisms (Table 11).

	Syntactic types					Semantic types					
	A	**B**	**C**	**D**	**E**	**a**	**b**	**c**	**d**	**e**	**f**
B1	0.42	0.40	0.38	0.67	0.29	0.40	0.58	0.52	0.45	0.32	0.25
C1	0.40	0.44	0.27	0.56	0.24	0.42	0.58	0.53	0.32	0.42	0.28
C2	0.40	0.42	0.31	0.60	0.29	0.30	0.53	0.42	0.28	0.32	0.32
C3	0.62	0.33	0.31	0.60	0.38	0.33	0.57	0.52	0.28	0.27	0.22
Avg. Acc.	0.46	0.40	0.32	0.61	0.30	0.36	0.57	0.50	0.33	0.33	0.27
SD	0.50	0.49	0.47	0.49	0.46	0.48	0.50	0.50	0.47	0.47	0.44
p-value	0.10	0.73	0.73	0.76	0.58	0.51	0.94	0.57	0.17	0.37	0.64

Table 11. The accuracy of error handling for different types of errors under each condition.

Furthermore, we found that different error-handling mechanisms did not significantly influence the accuracy of SQL query error handling at various difficulty levels (Table 12). These findings suggest that existing interaction mechanisms are not very effective for handling NL2SQL errors that state-of-the-art deep learning NL2SQL models make on complex datasets like Spider. We further discuss the reasons behind these results and their implications in the rest of Section 4.5 and Section 5.

F2: The error handling mechanisms do not significantly impact the overall time of completion. To study the impact of different error handling mechanisms on time usage, we analyzed the time of completion (ToC) of the query that was solved correctly by the participants. We used the same ANOVA test as applied in the previous analysis to test

	Difficulty levels		
	Easy	**Medium**	**Hard**
B1	0.64	0.64	0.21
C1	0.71	0.64	0.36
C2	0.79	0.71	0.36
C3	0.79	0.50	0.29
Avg. Acc	0.73	0.63	0.30
SD	0.45	0.49	0.46
p-value	0.81	0.71	0.83

Table 12. The error-handling of different difficulty levels under each condition

the mean difference among ToC using various error handling mechanisms (Table 10), no significant significance was found among the groups ($p = 0.52$).

Similarly, we analyzed the impact of different error-handling mechanisms on the selected error types. In general, the baseline method was more efficient in solving a task, while the conversational dialog system took more time compared with other methods. The results are shown in Table 13.

Additionally, the results of experiments on SQL queries of various levels of difficulty revealed differences among the error-handling mechanisms tested in the case of easy queries ($p = 0.04$). Specifically, direct editing was found to be the fastest method when the query was easy, followed by the explanation and example-based approach (C1), the explanation-based visualization approach (C2), and the conversational dialog system (C3).

	Syntactic types					**Semantic types**					
	A	**B**	**C**	**D**	**E**	**a**	**b**	**c**	**d**	**e**	**f**
B1	112.0	98.5	97.5	109.7	109.6	104.0	93.8	116.0	130.0	121.7	103.0
C1	103.3	103.8	91.1	104.6	101.7	96.1	103.9	110.1	107.0	97.9	83.4
C2	117.9	108.2	86.2	96.8	93.0	99.5	119.0	129.7	108.7	105.2	95.0
C3	129.2	110.3	123.6	125.8	133.8	118.4	125.0	148.6	116.2	125.6	125.0
Avg. ToC	115.6	105.2	99.6	109.2	109.5	104.5	110.4	126.1	115.5	106.3	101.6
SD	33.2	68.8	48.5	45.0	73.1	53.3	37.5	77.2	73.6	49.1	59.3
p-value	0.87	0.99	0.39	0.60	0.70	0.82	0.24	0.71	0.91	0.17	0.48

Table 13. The average ToC of different error types under each condition.

	Difficulty levels		
	Easy*	**Medium**	**Hard**
B1	31.8	124.4	133.6
C1	55.8	110.4	154.2
C2	79.0	110.5	199.1
C3	95.7	137.7	125.3
Avg. ToC	65.6	120.7	153.1
SD	50.9	96.2	97.7
p-value	0.04	0.60	0.39

Table 14. The average ToC of different difficulty levels under each condition. *statistically significant difference ($p < 0.05$)

F3: Users perform better on error types with fewer variants. We analyzed the impact of error types on task accuracy and ToC, and reported the results in Table 15. The results revealed that among the syntactic error types, `A: WHERE errors` and `E: GROUP BY errors` had high accuracy, while for semantic error types, `d: Value error` and `e: Condition error` had high accuracy. As shown in the error taxonomy (Table 3), value errors occur only in the WHERE clauses, and those errors usually require fewer steps to fix and have little relationship with the other syntactic parts in an SQL query. Similarly, condition errors such as `wrong sorting directions` and `wrong boolean operator (AND, OR, etc.)` are relatively independent components in a query. The better user performance on those error types may indicate that users face challenges in handling *semantically complicated* errors, such as joining tables and selecting columns from multiple tables, but are more successful in discovering and repairing error types where the error is more *local* (i.e., with little interdependency with other parts of the query). This conclusion is also evidenced in the user interview, which we will analyze in the following section.

Syntactic types	**Avg. Acc.** (μ = 0.53)	**SD** (μ = 0.50)	**Avg. ToC** (μ = 128.8)	**SD** (μ = 91.4)
A	0.56	0.51	132.3	105.5
B	0.48	0.50	147.2	90.4
C	0.53	0.51	128.2	75.6
D	0.53	0.47	111.5	55.5
E	0.55	0.51	125.1	72.4
Semantic types	**Avg. Acc.** (μ = 0.54)	**SD** (μ = 0.50)	**Avg. ToC** (μ = 123.1) **(N=26)**	**SD** (μ = 80.53)
a	0.47	0.49	123.0	67.4
b	0.54	0.51	123.3	68.2
c	0.50	0.51	128.7	68.7
d	0.61	0.47	118.1	96.5
e	0.60	0.49	116.7	83.2
f	0.51	0.51	128.1	66.8

Table 15. The average accuracy and ToC (in seconds) for different error types.

F4: The explanation- and example-based methods are more useful for non-expert users. When participants were asked to rate their preferences among the different interaction mechanisms (shown in Table 16), we found that the explanation- and example-based approach (C1) is the most preferred, while the explanation-based visualization approach (C2) was rated similarly to the baseline method (B1). In contrast, the conversational dialog system (C3) was generally rated as less useful than the others.

	Most useful	**2nd most useful**	**3rd most useful**	**least useful**
B1	7	4	9	6
C1	13	10	3	0
C2	5	8	9	4
C3	1	4	5	16

Table 16. The participants' ranked preferences for different error handling mechanisms

We found that the user's level of expertise significantly impacts their adoption rate of different error-handling mechanisms. The adoption rate measures when a mechanism was available, and how likely that a user will use the mechanism (instead of just using the baseline method) to handle the error. We calculated the adoption rate for each condition (C1, C2, and C3) for different levels of expertise by dividing the number of SQL queries in which the participant used the provided error-handling mechanism by the total number of queries provided with the corresponding mechanism in the participant's study session. The result is shown in Table 17.

Expertise levels	**C1** ($\mu = 0.74$)	**C2** ($\mu = 0.74$)	**C3** ($\mu = 0.41$)
Expert	0.53	0.43	0.41
Intermediate	0.84	0.90	0.44
Novice	0.86	0.88	0.38

Table 17. The adoption rate of each mechanism among different expertise levels

The primary factor contributing to the lower level of interest in using error handling mechanisms among expert participants under the experimental conditions was their ability to efficiently identify and repair errors independently. For example, P2 stated that *"It (the step-by-step execution function in C1) is very redundant and time-consuming to break down the SQL queries and execute the sub-queries, since most errors can be found at first glance."* Another reason why expert users were less interested in using the error handling mechanisms was that they were not confident in the intermediate results they provided. P3, for example, noted that *"Though the chatbot is capable of revising the erroneous SQL queries, I found it sometimes gives an incorrect answer and provides no additional clues for me to validate the new query."* Therefore, several expert participants chose to repair the original SQL query instead of validating and repairing the newly generated query.

The study also showed that the conversational dialog system (C3) was the least preferred mechanism among users at all levels of expertise. One reason for this is the relatively low accuracy of the model in recognizing user intents from the dialog and automatically repairing the errors in the query. For example, P3 stated that *"Though it sometimes predicts the correct query, for most of the times, the prediction is still erroneous."* In addition, the chatbot did not provide explanations for its suggestions, so users had to spend significant effort to validate and repair the newly generated SQL queries. Furthermore, while the chatbot allowed manual input from users to intervene in the prediction process, such as pointing out erroneous parts and providing correct answers, it often introduced new errors while predicting the SQL. As noted by P7: *"In one example, when I asked the chatbot to change the column name that was in SELECT, it somehow changes the column in JOIN as well."* As a result, many users quickly became frustrated after using it for a few SQL queries.

F5: The explanation- and example-based methods are more effective in helping users identify errors in the SQL query than in repairing errors. In the post-study questionnaire, we asked participants to evaluate the usefulness of each condition in terms of its ability to help (1) identify and (2) repair errors, respectively (Fig. 4). The results indicate that most of the participants found C1 to be effective in identifying incorrect parts of the SQL query, while half of them thought it was not useful for repairing errors. Meanwhile, a notable proportion of participants (12 out of 26) affirmed C2's effectiveness in identifying the errors, but it was helpful for repairing the errors. In terms of C3, a significant number of participants (16 and 18) had a negative perception of its effectiveness in both identifying and repairing errors within the SQL query.

Furthermore, we learned that the recursive natural language explanations might help reduce the understanding barrier for a long and syntactic-complicated SQL query. For example, P8 stated that *"By looking at the shorter sentences first at the beginning, I could finally understand the meaning that the original long sentence were trying to convey."* P17 also mentioned

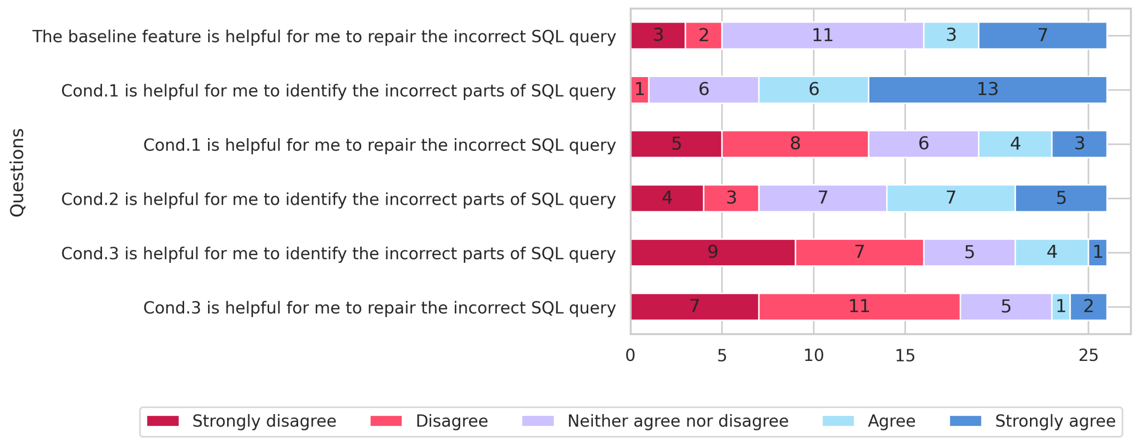


Fig. 4. The result of the post-study questionnaire

that: *"Those shorter sentences usually did not have complex grammatical structures and confusing inter-table relationships, so that the problems were easier to be spotted."* Additionally, executing the subquery and displaying the results were deemed helpful for localizing the erroneous parts in the original SQL query. For example, P23 stated: "When I noticed that the retrieved result was empty, I realized that some problems should exist in the current query." In terms of C2, participants affirmed the effectiveness of graph-based SQL visualization in helping them better understand the relationship between the syntactical components of a query. The learning barrier of this approach was also the lowest among all experimental conditions: users could view the connections to a table by simply clicking the widget in the canvas.

Then, we investigated why the participants were less satisfied with the effectiveness of repairing errors in a SQL query for C1. There were two main factors. First, the repair strategies supported by the error-handling mechanisms were limited. Specifically, participants could only *replace* the incorrect parts with their correct substitutions using the drop-down menu of entity mappings, but for queries that require the addition, deletion, or reorganization of clauses, users had to manually edit the query. This limitation led to frustration among participants and ultimately resulted in them not prioritizing using this error-handling mechanism for future tasks. Second, the current approach provided little assistance for users in validating their edits. As a result, one participant stated that: "I did not trust my own edits nor the suggested changes from the approach." (P20).

5 DISCUSSION AND DESIGN IMPLICATIONS

5.1 Improving NL2SQL model evaluations through the error taxonomy

Currently, the evaluation of NL2SQL models is mainly based on assessing their accuracy on large benchmark datasets. It is often unclear when and how the model fails. To address this limitation, our work exemplifies the value of the development of error taxonomies.

Our error taxonomy for NL2SQL models enabled new analysis. First, it allowed us to understand the types of syntactic and semantic errors that a particular NL2SQL model tends to make in addition to only the overall accuracy. This information can provide model developers with specific information to improve the model's robustness against certain error types. Second, this taxonomy allowed us to make fine-grained comparisons between models beyond the accuracy metrics. By comparing the error distributions of different models, we can identify not only the relative advantages of individual models but also the common errors that current models are prone to make.

5.2 Design opportunities for NL2SQL error handling mechanisms

The result of our empirical study suggests that existing error handling mechanisms do not perform as well on errors made by state-of-the-art deep-learning-based NL2SQL models on complex cross-domain tasks, despite the promising results reported in the respective evaluations of these mechanisms. We think the main reason could be that our study used a much more challenging dataset than what was used in prior studies. We used queries from Spider [63] (which is complex and cross-domain) that the state-of-the-art of NL2SQL models (instead of the earlier NL2SQL models, which would start to make errors on simpler SQL queries) failed on. The dataset used in our study more accurately represents realistic error scenarios that users encounter in natural language data queries. Here, we identified several design opportunities for more effective NL2SQL error-handling mechanisms.

5.2.1 Enabling effective mixed-initiative collaboration between users and error handling tools. Our findings indicate that the current error-handling tools for NL2SQL models do not provide sufficient feedback to users when they attempt to modify SQL queries. While existing error-handling mechanisms, such as the conversational dialog approach (C3), have focused on predicting correct modifications using static forms of user input, they have not adequately addressed the need for mechanisms to elicit useful human feedback to guide model prediction. For example, in C3, users provide input in the form of multiple-choice options for the recommended locations of potential errors, which was considered confusing and not useful by some participants, particularly when "none of the recommended options made sense" (P15) or "the errors existed in multiple places and cannot be fixed by only selecting one answer" (P24). Therefore, we suggest that future work should focus on the development of effective mixed initiative mechanisms that allow both users and error-handling tools to develop a mutual understanding of the model's current state of understanding and the user's intent.

5.2.2 Comprehending the generated queries and inspecting how queries operate on data complement each other. The results of the study suggested that, to support effective NL2SQL error handling for users, it is important to help users (1) interpret the meaning of the generated SQL query, untangle its structures, and explain how it corresponds to the user's NL query; and (2) inspect the behaviors of the query on example data and examine whether they match user intents and expectations. The two parts are interdependent on each other. In practice, the user's preferences for these different approaches may vary depending on their expertise. For example, in our study, non-users and novice SQL users appreciated the explanation-based visualization mechanism (in SQLVis [39]) and the NL explanations in step-by-step execution of the generated queries (in DIY [40]), because these mechanisms lower the barrier to understanding the generated SQL queries for users who are unfamiliar with SQL syntax and structures. This preference was also reflected in their use of different mechanisms in the study. Experienced SQL users, on the contrary, did not use mechanisms for explaining the meanings of the generated SQL queries as often. However, they found the entity mapping feature and the example tables (in DIY) useful for discovering NL2SQL errors.

5.2.3 Opportunities for adaptive strategies. Lastly, the results of our user study suggest that the most effective error handling strategy to use depends on many factors such as user expertise, query type, and possible errors types. For example, expert users may require less sense-making strategies (e.g., step-by-step NL explanation), while they may expect an intuitive execution result preview or an efficient validation of the updated answer. In contrast, intermediate or novice users may need more mix-initiative guides to facilitate error discovery and repair. Meanwhile, as discussed in

Section 4.5, the length, syntactical components, and potential error types of a query would result in different barriers to users when repairing errors. For example, for queries with more complicated syntactical structures, a visualization-based approach might be useful to reduce the barrier to understanding the structure of the query. Therefore, we recommend that future work in this area consider the development of adaptive error-handling strategies. An effective NL2SQL system could adapt its interface features, models, and interaction strategies according to the use case and context. Specifically, it could consider the semantic and syntactic characteristics of the query, whether the error is local (i.e., on a specific entity in the query) or global (i.e., regarding the overall query structure), and the user's preferences and level of expertise.

6 LIMITATIONS AND FUTURE WORK

The current study has several limitations. First, the distribution of error types in the SQL queries used in the study is unbalanced. As shown in Section 3.4.1, the error distributions show a large disparity in the number of errors for each type of error among the different models. Despite the fact that Spider is already a large-scale dataset, there were only a small number of example errors in some rare error types. Therefore, we must exclude these types of errors in our analysis. In the future, by recruiting more participants and collecting more data, we can include these categories of errors in the analysis.

Second, despite that we chose three representative state-of-the-art NL2SQL models in our analysis, there are several other promising NL2SQL models that we did not include in our analysis, mostly due to the lack of open-source implementation of the engineering challenges in adapting them for our pipeline. In addition, all the models used in our study are "black-box" models that do not provide much transparency into the process by which NL instructions are parsed and SQL queries are synthesized. Interactive models [14, 53], on the other hand, provide the transparency that could allow additional error handling mechanisms such as modifying the intermediate results of the model predictions. In future work, we will expand the scope of our research to include additional types of representative NL2SQL models.

Lastly, while the example SQL queries were real erroneous queries made by NL2SQL models on realistic databases and natural language queries, the setting of our study is still quite artificial, lacking the real-world task context in the actual usage scenarios of NL2SQL systems. In the future, it will be useful to study user error handling behaviors through a field study to better understand the impact of task-specific contexts on user behavior and the effectiveness of user handling of NL2SQL errors.

7 CONCLUSION

In this paper, we presented (1) an empirical study to understand the error types in the SQL query generated by NL2SQL models; and (2) a controlled user experiment with 26 participants to measure the effectiveness and efficiency of representative NL2SQL error handling mechanisms. The error taxonomy summarizes 48 error types and revealed their descriptive statistics. The results of the user experiment revealed challenges and limitations of existing NL2SQL error handling mechanisms on errors made by state-of-the-art deep-learning-based NL2SQL models on complex cross-domain tasks. Based on the results, we identified several research opportunities and design implications for more effective and efficient mechanisms for users to discover and repair errors in natural language database queries.

ACKNOWLEDGMENTS

This research was supported in part by an AnalytiXIN Faculty Fellowship, an NVIDIA Academic Hardware Grant, a Google Cloud Research Credit Award, and NSF grant CCF-2211428. Any opinions, findings, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Mohammad Aliannejadi, Manajit Chakraborty, Esteban Andrés Ríssola, and Fabio Crestani. 2020. Harnessing evolution of multi-turn conversations for effective answer retrieval. In *Proceedings of the 2020 Conference on Human Information Interaction and Retrieval*. 33–42.

[2] James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. Plow: A collaborative task learning agent. In *AAAI*, Vol. 7. 1514–1519.

[3] James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. 1996. A Robust System for Natural Spoken Dialogue. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics* (Santa Cruz, California) *(ACL '96)*. Association for Computational Linguistics, USA, 62–70. https://doi.org/10.3115/981863.981872

[4] Axel Antoine, Sylvain Malacria, Nicolai Marquardt, and Géry Casiez. 2021. Interaction Illustration Taxonomy: Classification of Styles and Techniques for Visually Representing Interaction Scenarios. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–22.

[5] Christopher Baik, Hosagrahar V Jagadish, and Yunyao Li. 2019. Bridging the semantic gap with SQL query logs in natural language interfaces to databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 374–385.

[6] Gagan Bansal, Besmira Nushi, Ece Kamar, Walter S Lasecki, Daniel S Weld, and Eric Horvitz. 2019. Beyond accuracy: The role of mental models in human-AI team performance. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, Vol. 7. 2–11.

[7] Jonathan Berant, Daniel Deutch, Amir Globerson, Tova Milo, and Tomer Wolfson. 2019. Explaining queries over web tables to non-experts. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1570–1573.

[8] Sonia Bergamaschi, Francesco Guerra, Matteo Interlandi, Raquel Trillo Lado, Yannis Velegrakis, et al. 2013. QUEST: a keyword search system for relational data based on semantic and machine learning techniques. (2013).

[9] Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-SQL parsing. *arXiv preprint arXiv:1905.06241* (2019).

[10] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[11] Felipe Costa, Sixun Ouyang, Peter Dolog, and Aonghus Lawlor. 2018. Automatic generation of natural language explanations. In *Proceedings of the 23rd international conference on intelligent user interfaces companion*. 1–2.

[12] Upol Ehsan, Brent Harrison, Larry Chan, and Mark O Riedl. 2018. Rationalization: A neural machine translation approach to generating natural language explanations. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. 81–87.

[13] Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your parser: Interactive text-to-SQL with natural language feedback. *arXiv preprint arXiv:2005.02539* (2020).

[14] Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo A. Ramos, and Ahmed Hassan Awadallah. 2021. NL-EDIT: Correcting Semantic Parse Errors through Natural Language Interaction. In *NAACL*.

[15] Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin* 76, 5 (1971), 378.

[16] Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R Woodward, John Drake, and Qiaofu Zhang. 2021. Natural SQL: making SQL easier to infer from natural language specifications. *arXiv preprint arXiv:2109.05153* (2021).

[17] Tong Gao, Mira Dontcheva, Eytan Adar, Zhicheng Liu, and Karrie G Karahalios. 2015. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th annual acm symposium on user interface software & technology*. 489–500.

[18] Simret Araya Gebreegziabher, Zheng Zhang, Xiaohang Tang, Yihao Meng, Elena Glassman, and Toby Jia-Jun Li. 2023. PaTAT: Human-AI Collaborative Qualitative Coding with Explainable Interactive Rule Synthesis. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. ACM.

[19] Barbara Grosz. 1983. Team: A transportable natural language interface system. In *Proceedings of the Conference on Applied Natural Language Processing (1983)*. Association for Computational Linguistics.

[20] 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. *arXiv preprint arXiv:1905.08205* (2019).

[21] Izzeddin Gür, 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)*. 1339–1349.

[22] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349* (2020).

[23] Junyang Huang, Yongbo Wang, Yongliang Wang, Yang Dong, and Yanghua Xiao. 2021. Relation Aware Semi-autoregressive Semantic Parsing for NL2SQL. *arXiv preprint arXiv:2108.00804* (2021).

[24] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760* (2017).

[25] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.

[26] Klaus Krippendorff. 2011. Computing Krippendorff's alpha-reliability. (2011).

[27] Shaopeng Lai, Qingyu Zhou, Jiali Zeng, Zhongli Li, Chao Li, Yunbo Cao, and Jinsong Su. 2022. Type-Driven Multi-Turn Corrections for Grammatical Error Correction. *arXiv preprint arXiv:2203.09136* (2022).

[28] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. *Research methods in human-computer interaction*. Morgan Kaufmann.

[29] Dongjun Lee. 2019. Clause-wise and recursive decoding for complex and cross-domain text-to-SQL generation. *arXiv preprint arXiv:1904.08835* (2019).

[30] Michael Levandowsky and David Winter. 1971. Distance between sets. *Nature* 234, 5323 (1971), 34–35.

[31] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, HV Jagadish, and Mirek Riedewald. 2020. QueryVis: Logic-based diagrams help users understand complicated SQL queries faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2303–2318.

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

[33] Fei Li and Hosagrahar V Jagadish. 2014. NaLIR: An Interactive Natural Language Interface for Querying Relational Databases. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 709–712. https://doi.org/10.1145/2588555.2594519

[34] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6038–6049. https://doi.org/10.1145/3025453.3025483

[35] Toby Jia-Jun Li, Jingya Chen, Haijun Xia, Tom Michael Mitchell, and Brad A. Myers. 2020. Multi-Modal Repairs of Conversational Breakdowns in Task-Oriented Dialogs. *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (2020).

[36] Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M. Mitchell, and Brad A. Myers. 2019. PUMICE: A Multi-Modal Agent that Learns Concepts and Conditionals from Natural Language and Demonstrations. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST 2019)*. ACM. https://doi.org/10.1145/3332165.3347899

[37] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *arXiv preprint arXiv:2012.12627* (2020).

[38] 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

[39] Daphne Miedema and George Fletcher. 2021. SQLVis: Visual Query Representations for Supporting SQL Learners. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE.

[40] Arpit Narechania, Adam Fourney, Bongshin Lee, and Gonzalo Ramos. 2021. DIY: Assessing the correctness of natural language to sql systems. In *26th International Conference on Intelligent User Interfaces*. 597–607.

[41] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*. 141–147.

[42] Laura Rieger and Lars Kai Hansen. 2020. A simple defense against adversarial attacks on heatmap explanations. *arXiv preprint arXiv:2007.06381* (2020).

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

[44] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Özcan. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1209–1220.

[45] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. 2017. Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296* (2017).

[46] Mark S. Schlager and William C. Ogden. 1986. A cognitive model of database querying: a tool for novice instruction. In *CHI '86*.

[47] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779

[48] Vidya Setlur, Sarah E Battersby, Melanie Tory, Rich Gossweiler, and Angel X Chang. 2016. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th annual symposium on user interface software and technology*. 365–377.

[49] Huda Salim Al Shuaily and Karen Vera Renaud. 2016. A Framework for SQL Learning: Linking Learning Taxonomy, Cognitive Model and Cross Cutting Factors. *World Academy of Science, Engineering and Technology, International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering* 10 (2016), 3095–3101.

[50] Alkis Simitsis and Yannis Ioannidis. 2009. DBMSs should talk back too. *arXiv preprint arXiv:0909.1786* (2009).

[51] Arjun Srinivasan and John Stasko. 2017. Orko: Facilitating multimodal interaction for visual exploration and analysis of networks. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 511–521.

[52] 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*. 177–186.

[53] 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 *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval* (Ann Arbor, MI, USA) *(SIGIR '18)*. Association for Computing Machinery, New York, NY, USA, 855–864. https://doi.org/10.1145/3209978.3210013

[54] Valentin Tablan, Danica Damljanovic, and Kalina Bontcheva. 2008. A natural language query interface to structured information. In *European Semantic Web Conference*. Springer, 361–375.

[55] Ningzhi Tang, Meng Chen, Zheng Ning, Aakash Bansal, Yu Huang, Collin McMillan, and Toby Jia-Jun Li. 2023. An Empirical Study of Developer Behaviors for Validating and Repairing AI-Generated Code. In *13th Annual Workshop at the Intersection of PL and HCI (PLATEAU 2023)*.

[56] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[57] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942* (2019).

[58] Xiaxia Wang, Sai Wu, Lidan Shou, and Ke Chen. 2021. An Interactive NL2SQL Approach with Reuse Strategy. In *International Conference on Database Systems for Advanced Applications*. Springer, 280–288.

[59] David H.D. Warren and Fernando C.N. Pereira. 1982. An Efficient Easily Adaptable System for Interpreting Natural Language Queries. *American Journal of Computational Linguistics* 8, 3-4 (1982), 110–122. https://aclanthology.org/J82-3002

[60] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. SQLizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 1–26.

[61] 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. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 5447–5458. https://doi.org/10.18653/v1/D19-1547

[62] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314* (2020).

[63] 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

[64] Tianyi Zhang, Zhiyang Chen, Yuanli Zhu, Priyan Vaithilingam, Xinyu Wang, and Elena L Glassman. 2021. Interpretable program synthesis. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–16.

[65] Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L Glassman. 2020. Interactive program synthesis by augmented examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 627–648.

[66] Zheng Zhang, Ying Xu, Yanhao Wang, Bingsheng Yao, Daniel Ritchie, Tongshuang Wu, Mo Yu, Dakuo Wang, and Toby Jia-Jun Li. 2022. StoryBuddy: A Human-AI Collaborative Chatbot for Parent-Child Interactive Storytelling with Flexible Parental Involvement. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 218, 21 pages. https://doi.org/10.1145/3491102.3517479

[67] Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded Adaptation for Zero-shot Executable Semantic Parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 6869–6882. https://doi.org/10.18653/v1/2020.emnlp-main.558

[68] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* (2017).

[69] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. 2018. Interpretable basis decomposition for visual explanation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 119–134.

[70] Xiangyang Zhou, Lu Li, Daxiang Dong, Yi Liu, Ying Chen, Wayne Xin Zhao, Dianhai Yu, and Hua Wu. 2018. Multi-turn response selection for chatbots with deep attention matching network. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1118–1127.