Insights into Natural Language Database Query Errors: From Attention Misalignment to User Handling Strategies

ZHENG NING*, University of Notre Dame, Notre Dame, USA
YUAN TIAN*, Purdue University, West Lafayette, USA
ZHENG ZHANG, University of Notre Dame, Notre Dame, USA
TIANYI ZHANG, Purdue University, West Lafayette, USA
TOBY JIA-JUN LI, University of Notre Dame, Notre Dame, USA

Querying structured databases with natural language (NL2SQL) has remained a difficult problem for years. Recently, the advancement of machine learning (ML), natural language processing (NLP), and large language models (LLM) have led to significant improvements in performance, with the best model achieving ~85% percent accuracy on the benchmark Spider dataset. However, there is a lack of a systematic understanding of the types, causes, and effectiveness of error-handling mechanisms of errors for erroneous queries nowadays. To bridge the gap, a taxonomy of errors made by four representative NL2SQL models was built in this work, along with an in-depth analysis of the errors. Second, the causes of model errors were explored by analyzing the model-human attention alignment to the natural language query. Last, a within-subjects user study with 26 participants was conducted to investigate the effectiveness of three interactive error-handling mechanisms in NL2SQL. Findings from this paper shed light on the design of model structure and error discovery and repair strategies for natural language data query interfaces in the future.

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

Additional Key Words and Phrases: Empirical study, human-computer interaction, error handling, database systems

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 [64, 68]. 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, scientific research, etc. 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 map the user's natural language query into a formal data query language such as SQL (NL2SQL). Such NL interfaces have shown the potential to lower the bar for data querying and support flexible data exploration for end users [5, 21, 75, 83].

*These authors contributed equally to this work.

**This work extends our previous research [53] published at IUI with new experiments on large language models (GPT-4) and a root cause analysis of the model attention and human attention.

Authors' addresses: Zheng Ning, University of Notre Dame, Notre Dame, IN, USA, zning@nd.edu; Yuan Tian, Purdue University, West Lafayette, IN, USA, tian211@purdue.edu; Zheng Zhang, University of Notre Dame, Notre Dame, IN, USA, zzhang37@nd.edu; Tianyi Zhang, Purdue University, West Lafayette, IN, USA, tianyi@purdue.edu; Toby Jia-Jun Li, University of Notre Dame, Notre Dame, IN, USA, toby.j.li@nd.edu.

ACM 2160-6455/2024/3-ART
https://doi.org/10.1145/3650114

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 [90], 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. In the past three years, advances in deep learning and large language models have brought us closer than ever to achieving useful performance on this important task—with the use of state-of-the-art end-to-end models such as [23, 24, 31, 57], the accuracy quickly increased to about 85%. However, the development in model performance appears to have stagnated in the 85% range recently, suggesting a bottleneck in model-only methods for NL2SQL.

This work focuses on the flip side of the 85% accuracy—the 15% erroneous queries. We started by understanding *"What errors current NL2SQL models make."* Then, we investigate *"How NL2SQL models made these errors"* and *"How users handle these errors"* with different studies.

We first performed a comprehensive analysis of SQL errors made by representative state-of-the-art NL2SQL models and developed an axial taxonomy of those errors. We reproduced four representative high-performing models with various structures from the Spider leaderboard*—DIN-SQL + GPT-4 [58], SmBop + GraPPa (SmBop) [61], BRIDGE v2 + BERT (BRIDGE) [47], and GAZP + BERT (GAZP) [95]. For each model, we collected all model-generated queries for the Spider dataset whose execution results varied from the ground truth results. Four authors conducted multiple rounds of qualitative coding and refinement on these errors to derive a taxonomy of the errors. The error analysis reveals that, despite having different model architectures and performance, NL2SQL errors originate from a common set of queries and demonstrate a similar distribution across various error types.

Given the error distribution, we further investigate the potential reasons behind the error. Inspired by recent work [6, 32, 46], which showed that attention alignment can improve the performance in code summarization, machine translation, and visual feature representation, we hypothesize that SQL errors generated by NL2SQL models derive from the misalignment between the model's attention and human's attention toward the natural language query. To validate this hypothesis, two authors (SQL experts) manually annotated important words (*human attention*) in the NL queries when they try to understand the query. The model-focused words (*model attention*) were obtained by calculating the weight of each word that contributed to the model's prediction using a perturbation-based method [48]. The attention alignment was measured by computing the overlap between the human-focused words and the model-focused words. The results showed a significant difference in attention alignment between erroneous queries and correctly predicted queries, implying that NL2SQL errors are highly correlated with attention misalignment. The findings suggest the promise of future work in aligning model attention with human attention to improve model performance.

To support NL2SQL models in real-world deployment, it is also important to provide effective error-handling mechanisms for users, which enable human users to discover and repair errors. In both human-computer interaction (HCI) and natural language processing (NLP) communities, we have seen efforts using different approaches. In general, there are three representative paradigms. First, following the *task decomposition* paradigm, strategies such as DIY [52] decompose a generated SQL statement into a sequence of incremental subqueries and provide step-by-step explanations to help users identify errors. Second, based on query visualization, approaches such as QueryVis [39], QUEST [9], and SQLVis [51] seek to improve user understanding of the generated SQL statements by visualizing the structures and relations among entities and tables in a query. Third, using conversational agents, works such as [29, 84, 88] implement chatbots to communicate the model's current state with users and update the results with user feedback through dialogs. These interactive approaches help the model and humans to synchronize their attention. Consequently, humans gain confidence in the decisions generated by the model, while the models can make better decisions under human guidance.

*https://yale-lily.github.io/spider

Although these approaches were shown to be useful in different contexts in individual evaluations, it is unclear how effective each approach is for users with various SQL expertise. Furthermore, as most of these methods were evaluated with only simple NL2SQL errors, it is unclear how well they will perform on errors made by state-of-the-art NL2SQL models on complex datasets such as Spider. Therefore, we selected three representative models and conducted a controlled user study ($N = 26$) to investigate the effectiveness and efficiency of representative error discovery and repair approaches. Specifically, we selected (i) an 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 [52]), (ii) an explanation-based SQL visualization approach (SQLVis [51]), and (iii) a conversational dialog approach [88]. 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 NL query interfaces.

To conclude, this paper presents the following four 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 comprehensive analysis that compared model attention to human attention. The result shows that NL2SQL errors are highly correlated with attention misalignment between humans and models.
- 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 database (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 [27, 85] or mapping rules [5, 40, 56, 62, 87]. 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 [34]. On the contrary, rule-based methods generally assume that there is a one-to-one correspondence between the words in q_{nl} and a subset of database keywords/entities [34]. 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 [34]. Second, the coverage of these methods is limited to a definite set of semantically tractable natural language queries [34, 56].

The recent development of deep learning (DL) based methods aims to achieve flexible NL2SQL translation through a data-driven approach [10, 28, 33, 47, 61, 95, 97]. From large-scale datasets, DL-based models learn to interpret NL queries conditioned on a relational DB via SQL logic [47]. Most NL2SQL models use the encoder-decoder architecture [47, 95, 97], 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 [47, 82] and pre-training techniques [30, 65, 89] 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 [61] and the LSTM-based pointer-generator [47]. However, those

DL-based models are usually "black-boxes" due to the lack of explainability [34]. 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 [90] and WikiSQL [97]. 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 [18, 52]. 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 [18, 69, 72, 77, 88]. For example, DIY [52] and STEPS [77] show step-by-step NL explanations and query results by applying templates to subqueries, helping users understand SQL output in an incremental way; similarly, SOVITE [44] 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; NaLIR [41] explains the mapping relations between entities in the input query and those in the database schema; Ioannidis et al. [69] 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 [8, 9, 39, 51]. For example, QueryVis [39] produces diagrams of SQL queries to capture their logical structures; QUEST [9] visualizes the connection between the matching entities from the input query and their correspondences in the database schema; SQLVis [51] introduced visual query representations to help SQL users understand the complex structure of SQL queries and verify their correctness.

Most of the previous work employed direct manipulation to repair and disambiguate queries. NaLIR [41], DIY [52] and DataTone [25] allow users to modify the entity mappings through drop-downs; Eviza [66] and Orko [71] 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 [88] maintains the state of the current parsing process and asks for human feedback to improve SQL translation through human-AI conversations. Elgohary et al. [18, 19] 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 [63]. Although there are some attempts to explain the state of the AI model using methods such as heatmap [60, 98], search traces [92], and natural language explanations [13, 17], 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*" [7]. Lastly, error handling usually requires multiple turns of interactions [36, 44]. 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, 99].

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 [76, 79, 93], data annotation [26], QA generation [94], interactive task learning [43, 45]), 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 four representative NL2SQL models and the corresponding error analysis. The structure of this section is as follows. Section 3.1 introduces the models we selected for analyzing the erroneous SQL queries. We also discussed the discrepancy in the workflow of different models. Section 3.2 summarizes the methodology used for building the dataset; Section 3.3 explains the axial and iterative coding procedure we used to derive the error taxonomy; Section 3.4 describes the developed taxonomy of NL2SQL errors; Section 3.5 presents an analysis of erroneous queries in the dataset based on the taxonomy.

3.1 Model selection

We selected four representative NL2SQL models from the official Spider leader board, the information of which is shown in Table 1.

While most models generate the SQL query in a top-down decoding procedure, SmBop (M1) improves the speed and accuracy by adopting a bottom-up structure. Specifically, it gradually combines smaller components to form a syntactically larger SQL component. BRIDGE (M2) is a sequential architecture that models the dependencies between natural language queries and relational databases. It combines a BERT-based [15] encoder with a sequential pointer-generator for end-to-end cross-DB NL2SQL semantic parsing. In comparison, GAZP (M3) combines a semantic parser with an utterance generator. When given a new database, it synthesizes data and selects instances that are consistent with the schema to adapt an existing semantic parser to this new database. DIN-SQL+GPT-4 (M4) has the best accuracy among all four models. It improves the performance of the large language model (GPT-4 [55]) on text-to-SQL tasks by employing task decomposition, adaptive prompting, linking schema to prompt, and self-correction. All these models employ different NL2SQL task-solving strategies at different stages, including decoding (M1), encoding (M2), finetuning (M3), and task preprocessing (M4).

Model Index	Model Names	Err. queries	Retrained Acc.	Original Acc.
M1	SmBop [61]	431	81.2%	71.1%
M2	BRIDGE [47]	853	62.7%	68.3%
M3	GAZP [96]	1062	53.6%	53.5%
M4	DIN-SQL+GPT-4 [58]	304	86.7%	85.3%

Table 1. Descriptive statistics and the accuracy of each model we reproduced

	NL query	SQL query
Easy	What is the abbreviation for airline ``JetBlue Airways" ?	SELECT Abbreviation FROM AIRLINES WHERE Airline = ``JetBlue Airways"
Medium	What are the codes of countries where Spanish is spoken by the largest percentage of people?	SELECT CountryCode , MAX(Percentage) FROM countrylanguage WHERE language= ``Spanish" GROUP BY CountryCode
Hard	What are the first names of the students who live in Haiti permanently or have the cell phone number 09700166582?	SELECT T1.first_name FROM students AS T1 JOIN addresses AS t2 ON T1.permanent_address_id = T2.address_id WHERE T2.country = 'haiti' OR T1.cell_mobile_number = '09700166582'
Extra Hard	What is the series name and country of all TV channels that are playing cartoons directed by Ben Jones and cartoons directed by Michael Chang?	SELECT T1.series_name , T1.country FROM TV_Channel AS T1 JOIN cartoon AS T2 ON T1.id = T2.Channel WHERE T2.directed_by = 'Michael Chang' INTERSECT SELECT T1.series_name , T1.country FROM TV_Channel AS T1 JOIN cartoon AS T2 ON T1.id = T2.Channel WHERE T2.directed_by = 'Ben Jones'

Table 2. NL-SQL pairs with different difficulty levels in the Spider dataset

3.2 Erroneous queries dataset collection

We adopted the Spider [90] dataset to train and evaluate the models and 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. It consists of around 10,000 queries in natural language on multiple databases across different domains (e.g., "soccer", "college"). In the original Spider dataset, the difficulty of the queries is divided into four levels: "Easy", "Medium", "Hard", and "Extra Hard", depending on the complexity of their structures and the SQL keywords involved. We demonstrate an example NL-SQL pair for each difficulty level in Table 2. In this work, we focus only on the first three difficulty levels as state-of-the-art models have significantly lower accuracy on the "extra hard" queries — the best model we reproduced, DIN-SQL+GPT-4, only achieved less than 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 close to those in the original training and developing sets. In addition, we ensured that there was no overlap in the databases used between our training and testing sets. Table 3 shows the distribution of our training and test data compared to the original public Spider dataset. We re-trained model M1-M3 with their officially released code using our training set. Since M4 was using few-shot learning, we did not retrain this model separately. The models used different core structures and showed close to SOTA performance at the time we conducted the study.

The erroneous queries are those that have different execution results from the correct ones (with values). The queries generated by these models on the test set were manually analyzed to develop the taxonomy of SQL generation errors. Table 1 shows the total number of erroneous queries generated by each model. The accuracy of each model on our test set is close to the reported performance of these models on the private held-out test set, indicating that our reproduction of these models are consistent with the original implementations.

		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 3. Descriptive statistics of the original Spider dataset and our sampled dataset

3.3 The coding procedure

After curating the dataset of erroneous queries, we followed the established open, axial and iterative coding process [4, 11, 37] to develop a taxonomy of NL2SQL errors. The detail of the process is as follows.

3.3.1 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 [4, 11, 37] 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 finishing the first round of coding, the coded query pairs (the generated query and the ground truth) were put line by line in a shared spreadsheet. The annotators sat together to discuss the codes and reached a consensus on the preliminary version of the codebook.

3.3.2 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 [50] (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 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.

3.3.3 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, 20, 35].

3.3.4 The annotation interface. We implemented an error annotation system to annotate the erroneous queries. The front-end UI is shown in Figure 1. It consists of three components: (A): A query display section that presents the natural language query, the corresponding ground truth, and the model-generated SQL query. (B): An error annotation section where the annotator first decided which part(s) of the generated SQL is wrong by clicking on the corresponding selection button. 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 accept open feedback. The error types would be updated after each batch of coding described in Section 3.3.2. (C): A results display section. The tables involved in the pairs and the execution result were shown in this section to help annotators identify the error types.

3.4 The Taxonomy of NL2SQL Errors

Table 4 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

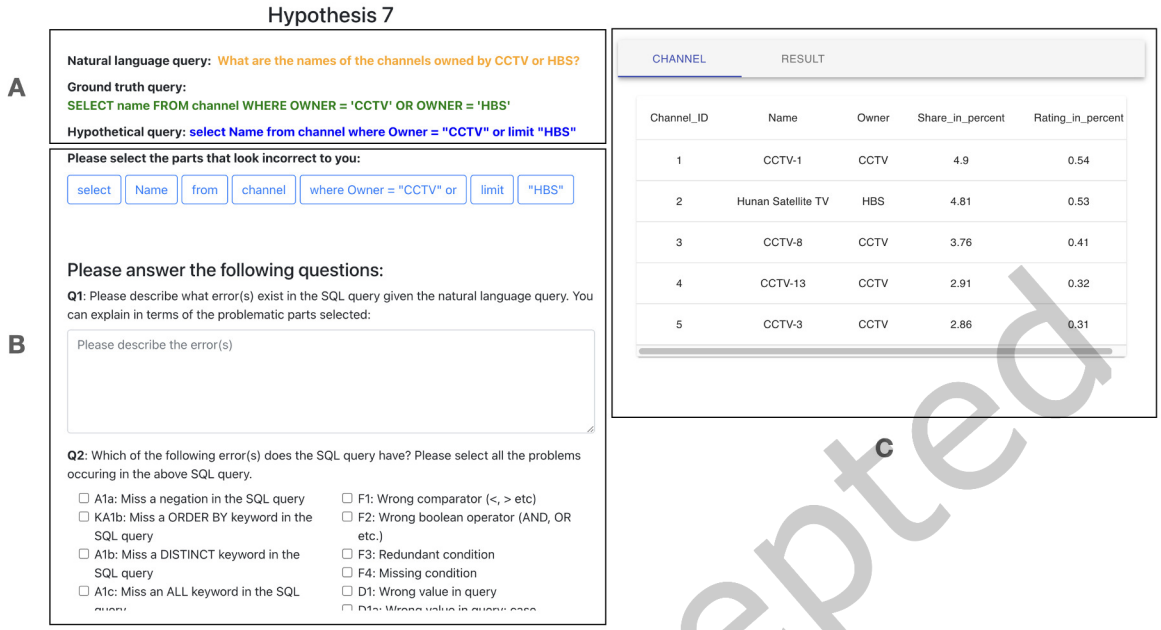

Fig. 1. The user interface that we used for NL2SQL error annotation

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 "`Miss a table to JOIN`" (Ba1) and "`JOIN the wrong table`" (Ba2). An erroneous query may also have multiple error types associated with it.

3.5 NL2SQL error analysis

Based on the error taxonomy, we further analyzed the erroneous queries to explore the following three questions:

RQ1 How are the erroneous queries distributed among different models? Do models tend to stumble on the same queries or make mistakes on distinct queries? Furthermore, for those overlapping erroneous queries, do models tend to make similar types of error on them or not?

RQ2 How do error types spread along the syntactic and semantic dimensions? How different are the distributions of error types among the three models?

RQ3 How far are the erroneous queries from their corresponding ground truths?

3.5.1 The distribution of erroneous queries among models. Figure 2 shows the overlap of erroneous queries among the best three models in a Venn diagram (DIN-SQL+GPT-4, SmBop, and BRIDGE). 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 1).

Furthermore, we found that there were 129 queries appeared in all four models, we sampled three of them and presented the queries and error types in Table 6. Additionally, we found 92.1% (280 out of 304) of DIN-SQL+GPT-4's; 82.4% (355 out of 431) of SmBop's; 84.4% (720 out of 853) of BRIDGE's; and 70.6% (750 out of 1062) of GAZP's

Error categories	Error types			SmBop	BRIDGE	GAZP	DIN-SQL+ GPT-4
Syntactic errors	A: WHERE error	Aa1:	Use a wrong table in WHERE	21	68	73	10
		Ab1:	Use a wrong column in WHERE	19	36	23	12
		Ac1:	Redundant WHERE clause	14	16	27	13
		Ac2:	Missing WHERE clause	15	21	61	7
		Ad1:	Other wrong value in WHERE clause	51	52	91	18
		Ad2:	Value case error in WHERE clause	62	69	82	36
		Ad3:	Value plurality error in WHERE clause	8	6	16	0
		Ad4:	Value synonym error in WHERE clause	35	40	45	35
		Ae1:	Wrong comparator (<, >, =, !=, etc)	8	13	14	3
		Ae2:	Wrong boolean operator (AND, OR etc.)	4	15	9	3
	B: JOIN error	Ba1:	Miss a table to JOIN	35	106	101	15
		Ba2:	JOIN the wrong table	24	89	78	3
		Bb1:	Use a wrong column in JOIN	13	79	69	7
		Bc1:	Redudant JOIN clause	17	82	113	36
	C: ORDER BY error	Cb1:	Use a wrong column to sort	3	26	26	9
		Cc1:	Miss a ORDER BY clause	12	22	20	4
		Cc2:	Redundant sorting	3	1	3	0
		Ce1:	Wrong sorting direction	6	27	23	15
	D: SELECT error	Da1:	Use a wrong table in SELECT	59	117	106	26
		Db1:	Return a wrong column in SELECT	21	56	78	33
		Db2:	Return a redundant column in SELECT	10	19	36	13
		Db3:	Miss returning column(s) in SELECT	20	34	59	11
		Df1:	Use wrong aggretation function	7	43	11	9
		Df2:	Miss aggregation function	5	19	14	6
	E: GROUP BY error	Eb1:	Use a wrong column in GROUP BY	6	10	18	12
		Ec1:	Miss a GROUP BY clause in the SQL query	10	33	47	8
		Ec2:	Redudant GROUP BY clause	6	7	16	9
	F: HAVING error	Fc1:	Miss HAVING clause	1	5	12	1
		Fc2:	Redundant HAVING clause	0	2	6	1
		Fe1:	Wrong condition in HAVING	1	2	3	6
	G: LIKE error	Gc1:	Miss LIKE clause	1	3	9	4
		Ge1:	Wrong LIKE condition	1	8	22	2
	H: LIMIT error	Hc1:	Redudant LIMIT clause	1	2	6	0
		Hc2:	Miss LIMIT clause	0	1	3	1
	I: INTERSECT error	Ie1:	Wrong INTERSECT condition	8	8	9	11
	J: DISTINCT error	Jc1	Miss a DISTINCT keyword	7	18	96	12
		Jc2	Redundant DISTINCT keyword	4	15	0	11
	K: EXCEPT error	Kc1	Wrong EXCEPT clause	14	27	24	13
	L: NOT error	Lc1	Miss NOT keyword	7	9	7	0
	M: UNION	Me1	Wrong UNION condition	9	9	8	10
Semantic errors	a: Table error			97	279	274	53
	b: Column error			81	237	251	94
	c: Miss/redundant Clause/keyword error			107	250	432	101
	d: Value error			153	162	230	82
	e: Condition error			37	82	88	49
	f: Aggregation function error			12	62	25	15

Table 4. The taxonomy of NL2SQL errors with the count of each error type for the models

incorrect queries also confounded other models, indicating that new models are making errors on a limited number of new queries each time. The statistics are also presented in Table 5. The results imply that ***different models tend to make errors on the same subset of queries in NL2SQL.***

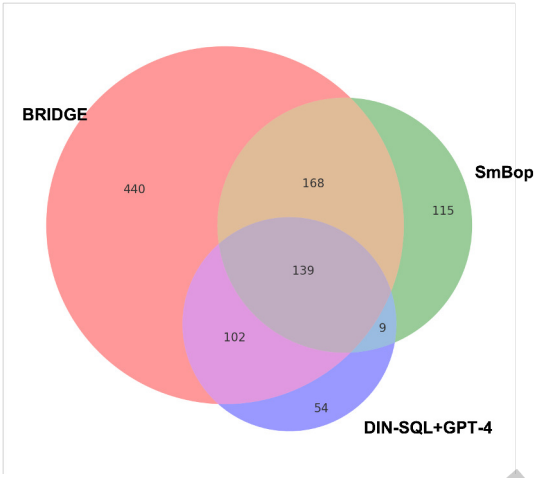

Fig. 2. The overlap of erroneous queries generated by DIN-SQL+GPT-4, SmBop, and BRIDGE

Model Names	Overlapped Queries Percentage
DIN-SQL+GPT-4	92.1%
SmBop	82.4%
BRIDGE	84.4%
GAZP	70.6%

Table 5. The percentage of erroneous queries for each model that also appeared in the other three models

$$Jaccard\ coefficient = \frac{|E_{DIN-SQL+GPT-4} \cap E_{SmBop} \cap E_{BRIDGE} \cap E_{GAZP}|}{|E_{DIN-SQL+GPT-4} \cup E_{SmBop} \cup E_{BRIDGE} \cup E_{GAZP}|} \quad (1)$$

To understand whether models make similar types of errors in those overlapped queries, for each query, we measured the similarity of the syntactic and semantic error types respectively among the models. Noticeably, there are multiple distance metrics such as Jaccard distance [38], Hamming distance, and Euclidean distance. Hamming distance usually works for sets with the same length, while Euclidean distance measures the distance between two points in an Euclidean space. In our case, these two different error sets are challenging to map to the space. Thus, we choose the Jaccard distance as it can be adapted to sets of different lengths and explicitly considers the difference of each error type in the set. The Jaccard coefficient is measured using Equation 1, where E_m means the set of error types that the model m made on the target query. For syntactic error types, 27.9% of overlapped queries have a Jaccard coefficient of 0 among the four models, which implies that the models did not all make the same syntactic error type in these queries. Regarding the semantic error types, 24.8% of these queries have a Jaccard coefficient of 0. On the other hand, only 7.8% of the overlapped queries have the same syntactic error type from the four models, and even fewer of them (6.2%) have the same semantic error type from all four 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 6 to illustrate the similarity and disparity of error types in the same queries.

3.5.2 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		2		3	
NL query	How many games are played for all students?		What are the different membership levels?		Find the package choice and series name of the TV channel that has high definition TV.	
Correct query	SELECT sum(gamesplayed) FROM Sportsinfo		SELECT count(DISTINCT level) FROM member		SELECT package_option, series_name FROM TV_Channel WHERE high_definition_TV = "yes"	
Model	**Generated query**	**Error types**	**Generated query**	**Error types**	**Generated query**	**Error types**
DIN-SQL +GPT-4	SELECT COUNT(GameID) FROM Plays_Games	Da1, Db1, Df1	SELECT DISTINCT Level FROM member	Df2	SELECT package_option, series_name FROM TV_Channel WHERE high_definition_TV = "Yes"	Ad2
SmBop	SELECT COUNT(*) , stuid FROM plays_games	Da1, Db1, Db2, Df1	SELECT Level FROM member	Df2, Jc1	SELECT package_option, series_name FROM TV_Channel WHERE high_definition_TV = 1	Ad1
BRIDGE	SELECT COUNT(*) FROM Plays_Games	Da1, Db1, Df1	SELECT DISTINCT level FROM member	Df2	SELECT package_option, series_name FROM TV_Channel WHERE high_definition_TV = "t"	Ad4
GAZP	SELECT count (*) FROM Plays_Games	Da1, Db1, Df1	SELECT level FROM member	Df2, Jc1	SELECT package_option, series_name FROM TV_Channel WHERE high_definition_TV = "definition"	Ad1

Table 6. Sampled erroneous NL-SQL pairs and their error types

(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 entire development set that has the corresponding syntax. (Table 7). 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 7). 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 of erroneous queries made by the model m. (Table 8). It measures the percentage of queries that contain a specific type of semantic error among all erroneous queries.

As shown in Table 7, the distributions of syntactic error type are similar among all four models. Note that a model can produce a query with multiple types of errors. Notably, the error percentages 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 4% of erroneous queries among all models, it has an error rate of more than 50% (i.e., when the correct query should 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 9.

Compared to syntactic errors, the distribution of semantic errors is more varied between models (Figure 8). We found that `d: Value error` and `a: Table error` are the most frequent error types for SmBop (35.5%) and BRIDGE (32.71%), respectively. While `c: Miss/redundant clause/keyword error` appeared most frequently in GAZP (40.68%) and DIN-SQL+GPT-4 (33.22%). It indicates that ***the semantic challenges of the investigated NL2SQL models are more varied than the syntactic challenges they faced.***

3.5.3 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 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. Before computing the distance, we first

Error type	Error Percentage				Error Rate			
	SmBop	BRIDGE	GAZP	DIN-SQL+ GPT-4	SmBop	BRIDGE	GAZP	DIN-SQL+ GPT-4
A: WHERE error	47.80%	35.05%	30.89%	42.43%	18.86%	27.38%	30.04%	11.81%
B: JOIN error	17.87%	28.14%	31.83%	19.74%	10.13%	31.58%	44.47%	7.89%
C: ORDER BY error	4.64%	7.39%	5.74%	9.87%	4.58%	14.42%	13.96%	6.86%
D: SELECT error	23.20%	25.79%	25.80%	30.92%	4.37%	9.62%	11.98%	4.11%
E: GROUP BY error	5.10%	5.86%	7.63%	9.54%	4.50%	10.22%	16.56%	5.93%
F: HAVING error	0.46%	1.06%	1.98%	2.63%	1.44%	6.47%	15.11%	5.76%
G: LIKE error	0.46%	1.29%	2.92%	1.97%	2.86%	15.71%	44.29%	8.57%
H: LIMIT error	0.23%	0.35%	0.85%	0.33%	0.41%	1.24%	3.73%	0.41%
I: INTERSECT error	1.86%	0.94%	0.85%	3.62%	18.60%	18.60%	20.93%	25.58%
J: DISTINCT error	2.55%	3.87%	9.04%	7.57%	4.78%	14.35%	41.74%	1.74%
K: EXCEPT error	3.25%	3.17%	2.26%	4.28%	20.29%	39.13%	34.78%	18.84%
L: NOT error	1.62%	1.06%	0.66%	0.00%	13.46%	17.31%	13.46%	0.00%
M: UNION error	2.09%	1.06%	0.75%	3.29%	56.25%	56.25%	50.00%	62.5%

Table 7. The error percentage and error rate of each syntactic error type

Error type	SmBop	BRIDGE	GAZP	DIN-SQL +GPT-4
a: Table error	22.51%	32.71%	25.80%	17.43%
b: Column error	18.79%	27.78%	23.63%	30.92%
c: Miss/redundant Clause/keyword error	24.83%	29.31%	40.68%	33.22%
d: Value error	35.50%	18.99%	21.66%	26.97%
e: Condition error	8.58%	9.61%	8.29%	16.12%
f: Aggregation function error	2.78%	7.27%	2.35%	4.93%

Table 8. The error percentage of each semantic error type

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%
DIN-SQL+GPT-4	UNION 62.5%	INTERSECT 25.58%	EXCEPT 18.84%	WHERE 11.81%	LIKE 8.57%

Table 9. The top 5 error-prone syntactic parts of a SQL query for the selected models

pre-process the ground truth and predicted queries for each pair to unify the SQL format. Specifically, we i): ignore the differences in the upper or lower case letters except for values; ii) extract the table names and the alias for each column used in the query and prefix the column names with the identified table names. Noticeably, there may be other cases where the predicted query does not need to be revised to exactly the form of the ground truth query in order to get the correct result; therefore, the Levenshtein distances we obtained may be larger than the actual ones.

Figure 3 shows the distribution of the Levenshtein distances of errors made by each model. It is worth noting that all four 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 four models can be fixed in a small number of edits. Especially for the best model we reproduced, DIN-SQL+GPT-4, 19.1% (58/304) of the erroneous queries can be fixed in one step. 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.

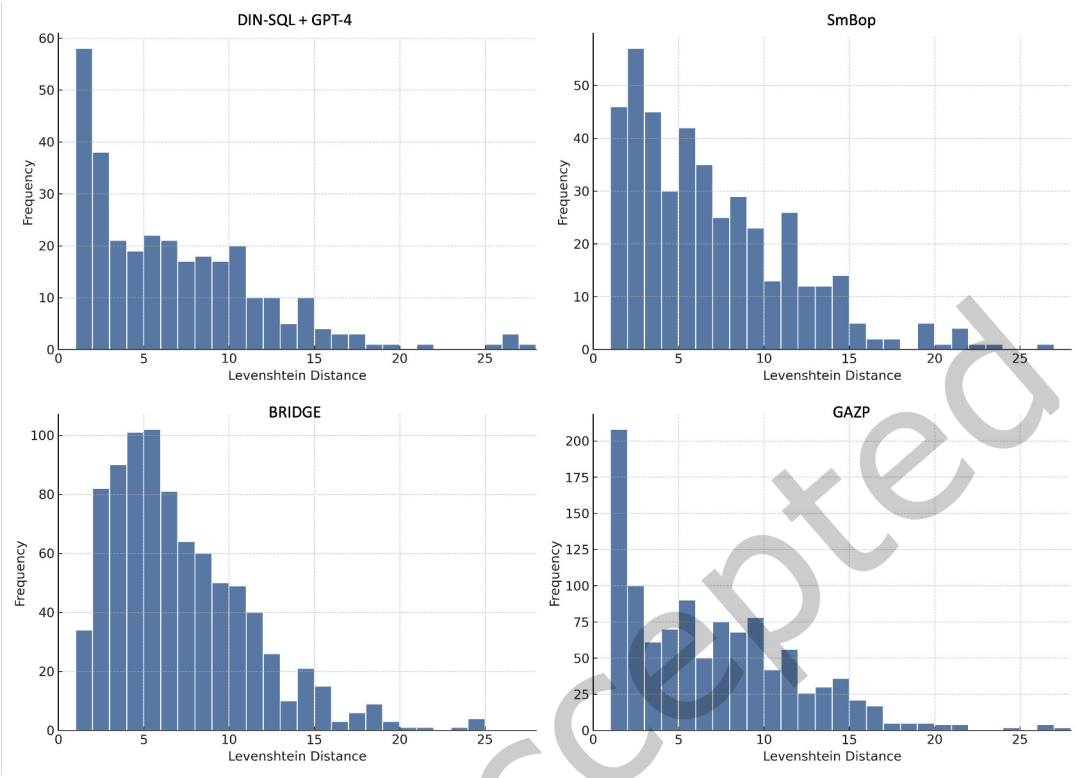

Fig. 3. The distribution of Levenshtein distances between erroneous queries and ground truth queries for each model

4 ANALYSIS ON NL2SQL HUMAN-MODEL ATTENTION ALIGNMENT

After understanding the error types and distribution in NL2SQL, we aim to further investigate the possible cause of SQL errors. Previous research [6, 32, 46] has shown that the discrepancy between model attention and human attention is correlated with poor performance in code summarization, machine translation, and visual feature representation. We hypothesize that such misalignment between model attention and human attention is also a source of error in NL2SQL. In other words, NL2SQL models return incorrect columns or tables in part because they do not pay attention to words that humans pay attention to. However, the lack of empirical research limits our understanding in this area.

To investigate whether and to what extent SQL errors derive from such attention misalignment, we compare human-labeled attention with a representative model's attention on NL queries.

4.1 Data preparation

4.1.1 Different attention calculation methods. There are different methods to obtain the model's attention. For example, some work [12, 22, 42, 80] leveraged the self-attention mechanism of transformer [81] to obtain attention, while some work [14, 67, 70, 74] used the gradients of the model predictions with respect to the input features to calculate the model's attention.

Perturbation-based methods follow a two-step process: they first mutate the input and then calculate the model's attention based on the differences in the output. For instance, LIME [59] generates a local explanation

by approximating the specific model predictions using a simpler model, such as a linear classifier. SHAP [49] improves on LIME by perturbing the input based on game theory and using the Shapely value to estimate the importance of different tokens. However, a limitation of both methods is that they often require a large number of perturbed samples to ensure an accurate estimation. Furthermore, LIME and SHAP only mutate an input by deleting tokens, which may significantly alter the meaning or structure of the input. To address this limitation, more recent perturbation-based methods opt to replace tokens with similar or semantically related tokens in the context [48, 86]. These methods often utilize a masked language model such as BERT [16] to predict similar or semantically related tokens to replace existing tokens in an input. In this study, we selected a perturbation-based method optimized by BERT. The reasons for this choice will be explained in the following paragraphs.

4.1.2 NL2SQL model selection. We selected SmBoP [61], which achieves the second-highest performance among the models we used in the previous study. Although DIN-SQL + GPT-4 [58] achieves the highest performance, GPT-4 is not open-source, therefore, we cannot directly calculate its attention based on self-attention or gradient. Additionally, by using the perturbation-based method, the GPT-4 API has to be called millions of times, which is not practical and affordable.

4.1.3 Attention calculation method selection. Methods leveraging either self-attention or gradient are not suitable for a model with multiple components other than a transformer [81] since attention or gradient does not represent the attention for the entire model. The decoding process of SmBoP starts by generating SQL components and gradually combines them from the bottom up to create a complete SQL statement. This process involves contextualization based on the transformer's self-attention mechanism and the use of predefined rules to filter out invalid queries.

Due to the complexity of SmBoP architecture, the attention in transformer headers or simply using gradients cannot accurately represent the attention of the entire model. Therefore, we choose the perturbation-based method, which regards the SmBoP model as a black box without considering the inner details of its architecture.

4.2 Human attention labeling

To obtain human attention on NL2SQL tasks, two annotators, who are proficient in SQL, conducted iterative refinements of attention annotation. We first randomly sampled 200 tasks from the Spider dataset, where 140 tasks can be correctly solved by the experiment model (SmBoP), and 60 tasks on which the model makes errors. We intentionally balanced the sampled tasks according to the performance of SmBoP (69.5% on the Spider test set).

Each iteration consisted of the following three steps. First, two annotators separately annotated a new sample batch of 25 tasks. For each task, the annotators reviewed the natural language (NL) question and the ground truth SQL query. Then each annotator individually identified important NL words (those that contribute to the query) and marked their attention weight as **1** while marking the attention weight of the remaining unimportant words (e.g., all, the, of) as **0**. Second, we computed the inter-rater reliability between annotators [50] (Fleiss' Kappa and Krippendorff's Alpha) at the end of each iteration. Lastly, annotators discussed how they judge the importance of a certain word. Annotators completed three refinement iterations until the human-labeled attention became stable and the inter-rater reliability scores were substantial. At the end of the final refinement iteration, Fleiss' Kappa was 0.67 and Krippendorff's Alpha was 0.58.

4.3 Measuring attention alignment between human and model

We measure the alignment between the attention of the models and the annotators using the *keyword coverage rate*. Specifically, we select the Top K model-focused words with the highest attention as $word_m$, where K equals

the number of keywords selected by human annotators. Then we calculate the percentage of human-labeled words $word_h$ covered in $word_m$, as shown in Equation 2.

$$rate = \frac{|word_m \cap word_h|}{|word_h|} \tag{2}$$

Nevertheless, the attention scores calculated by this method cannot be directly compared to words annotated by human labelers. This is because human labelers annotate attention on each individual word, while machine attention is calculated and distributed on specific tokenization of the model. For example, the word "apple" can be tokenized into two tokens: "ap" and "ple" through byte pair encoding. To bridge the gap, we developed a method to map model tokens back to individual words and recalculate the attention distribution. Suppose an input text includes N words $\{w_1, w_2, ..., w_n\}$, while the model's tokenizer splits the text into M tokens $\{t_1, t_2, ..., t_m\}$. We calculate the model's attention on ith word w_i as the sum of all tokens that overlap with w_i, as shown in Equation 4.

$$attention_{w_i} = \sum_{t_j \cap w_i \neq \emptyset} attention_{t_j} \tag{3}$$

4.4 Results

In this section, we aim to answer the following research questions:

RQ1 To what extent model attention is aligned with human attention?
RQ2 Can the attention misalignment explain the errorous queries generated by NL2SQL models?
RQ3 Which error types are highly related to the attention misalignment?

In our 200 sampled tasks from the Spider dataset, the average number of words in the NL queries is 12. Accordingly, we calculate the *Keyword Coverage Rate* by experimenting with different K (i.e., the top K words with the highest attention to be considered in the calculation) from 1 to 20. Since the value of K may exceed the number of words in the NL query, the actual number of words considered is equal to the minimum of K and the number of words in the NL query.

K	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Alignment	0.138	0.231	0.302	0.368	0.433	0.491	0.545	0.601	0.649	0.698
K	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Alignment	0.740	0.776	0.804	0.832	0.851	0.868	0.880	0.888	0.894	0.898

Table 10. Average Keyword Coverage Rate for all 200 tasks

F1: The model's attention partially aligns with human attention, which is consistent with the model's performance. Table 10 shows the average *Keyword Coverage Rate* for all 200 tasks in different settings of K. For example, when considering the top 12 words (the average number of words), there is an overlap of around 77.6% between human-focused and model-focused words. This result is consistent with the performance of SmBoP, which achieves 70% accuracy on the sampled dataset.

F2: Attention alignment is higher when the model correctly solves the task, suggesting that SQL errors are correlated with attention misalignment. To examine the relationship between attention alignment and the model's performance, we compare the *Keyword Coverage Rate* of correctly solved tasks with that of incorrectly

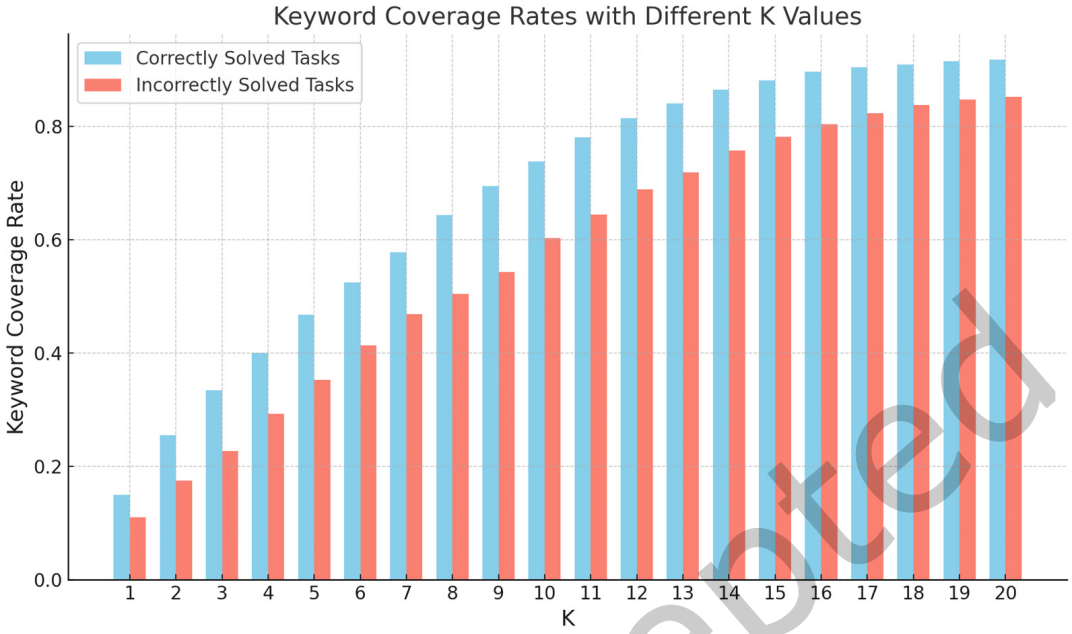

Fig. 4. The distribution of alignment for correctly and incorrectly solved tasks considering different numbers of keywords.

solved tasks. Figure 4 shows the average keyword coverage rate with different K values for correct and incorrect queries generated by SmBoP. When SmBoP generates a correct SQL query, the attention alignment is significantly higher than when it generates an erroneous SQL for all K values, with all p-values being less than 0.05.

Take $K = 6$ (half of the number of words) as an example, Figure 5 shows the comparison of attention alignment distributions between correctly and incorrectly generated queries. Each distribution approximately follows a Gaussian distribution. However, the mean of the distribution for erroneous queries is significantly lower than that of correct queries, with a p-value of $1.5e-4$. Furthermore, when generated queries are correct, all the rates are no more than 0.67, suggesting that a low attention alignment contributes to generating an erroneous SQL.

F3: Attention misalignment is not specific to a certain error type. To further investigate which types of errors are more correlated with attention misalignment, we labeled the error types of all 40 incorrectly solved tasks using the same procedure discussed in Section 3.3. Next, we calculated the average keyword coverage rate associated with each type of error. Specifically, for a certain error type, we summed up the attention alignment of all tasks that included that error. Then, we divided the summed attention alignment score by the number of tasks that included this error, as shown in Equation 2.

$$align_{E_i} = \frac{\sum_{E_i \in task_j} align_{task_j}}{\left|\{\text{task}_i \mid \text{task}_j \text{ contains error } E_i\}\right|} \tag{4}$$

Table 11 shows the average alignment for each type of error. The results indicate no significant difference in attention alignment across different types of error.

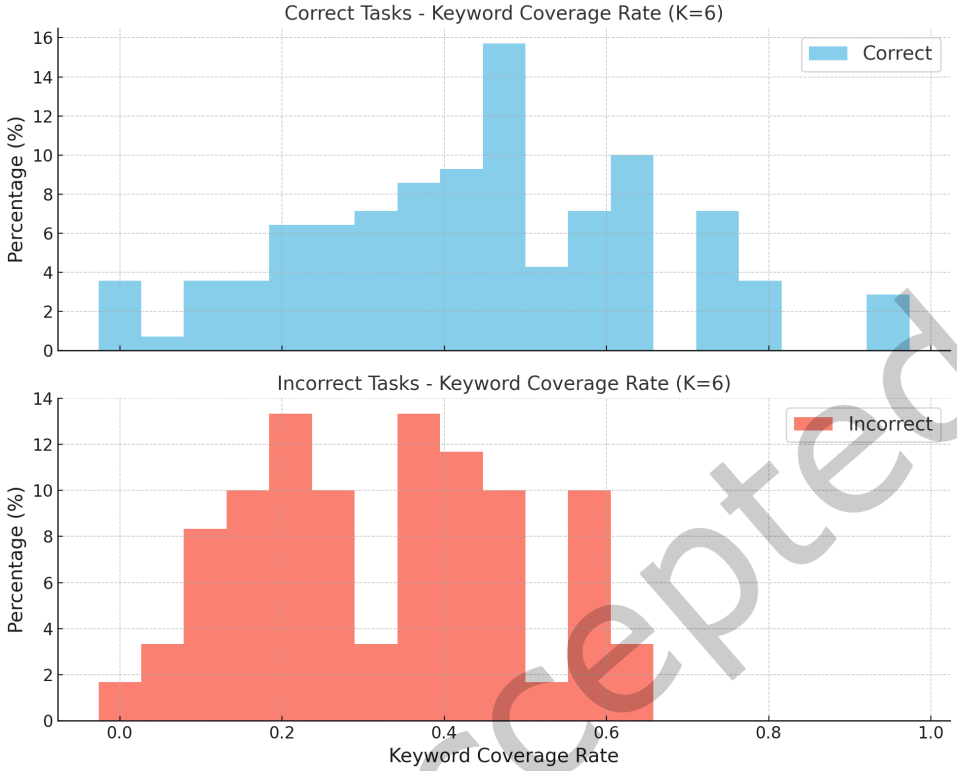

Fig. 5. A comparison of attention alignment between correctly and incorrectly solved tasks.

Table 11. Average attention alignment for different types of error.

Syntactic error type	*A*	*B*	*C*	*D*	*E*	*F*	*G*
Attention alignment	0.26	0.32	0.22	0.30	0.25	0.27	0.34
Syntactic error type	*H*	*I*	*J*	*K*	*L*	*M*	
Attention alignment	0.19	0.28	0.33	0.24	0.21	0.26	

5 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 [9, 29, 39, 51, 52, 84, 88], regardless of users' domain expertise of using SQL language. To understand the performance and usage of these mechanisms by users, we conducted a controlled user study to examine the effectiveness of different error discovery and repair mechanisms for NL2SQL[†]. Specifically, we investigated the following research questions.

[†]The protocol of the study has been reviewed and approved by the IRB at our institution.

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 [52])
Exp. Cond. #2	Graph-based SQL query visualization (SQLViz [51])
Exp. Cond. #3	Conversational dialog system (MISP [88])

Table 12. The list of conditions used in the user study

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?

5.1 Experiment conditions

In this study, we used four conditions shown in Table 12. 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 [52]) 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 [51]). 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 [88]). It allows users to correct an erroneous SQL query through multiple rounds of conversation in natural language (Table 12). We replicated the core functionalities of the DIY mechanism used in experimental condition #1 as described in the paper [52] because the official source code was not publicly released. For experimental condition #2, we used the official implementation[‡] 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[§] for the study.

5.2 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 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.

[‡]https://github.com/Giraphne/sqlvis
[§]https://www.npmjs.com/package/react-chatbot-kit

5.3 Study procedure

In our study, each participant experienced the four conditions described in Section 5.1. 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 [11, 37], 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.

5.4 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.

5.5 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

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

differences between the different error handling methods. The average accuracy and standard deviation among the participants are shown in Table 13.

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 7. 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 14).

	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 14. 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 15). 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 5.5 and Section 6.

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

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 the mean difference among ToC using various error handling mechanisms (Table 13), 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 16.

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 16. 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 17. 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 18. 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 4), 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 18. 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 19), 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 19. 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 20.

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

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 20. The adoption rate of each mechanism among different expertise levels

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. 6). 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 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

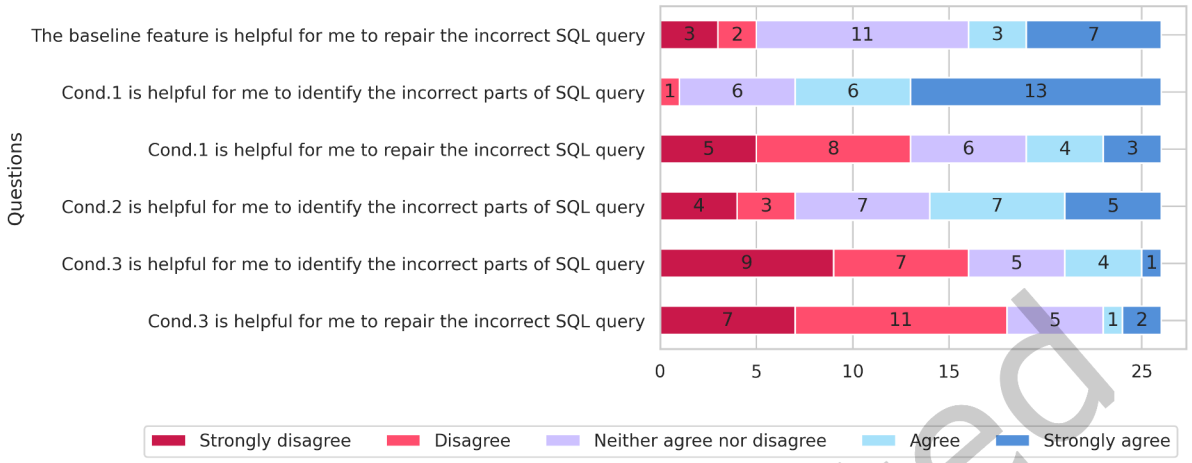

Fig. 6. The result of the post-study questionnaire

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).

6 DISCUSSION AND IMPLICATIONS

6.1 Improving NL2SQL model evaluations through the error taxonomy

Currently, the evaluation of NL2SQL models emphasizes their accuracy from benchmark datasets. Though it is fair and effective in indicating the overall performance of the model, it fails in evaluating the model at a more fine-grained level, which impedes the development of error-handling strategies and the model's real-world application.

The error taxonomy we contributed helps us 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, thereby developing more accurate NL2SQL models. Future work can focus on the technical solutions on how to design better NL2SQL models based on the taxonomy. 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.

Our work delves deep into the errors of the models, revealing that though model architectures and performances differ, they all exhibit high error rates in particular error types. On the other hand, some models, though have a relatively low overall accuracy, are capable of handling particular types of errors. Future work could focus on studying one of those high error-rate error types to increase the model performance.

6.2 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 [90] (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.

6.2.1 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.

6.2.2 Implementing interactive approaches based on attention alignment. Our study and analysis on attention have proved the correlation between erroneous queries and attention misalignment, suggesting that a potential way of designing an error-handling mechanism is to enable users to correct the misalignment. However, a majority of existing work focuses on automated attention alignment in the decoding process of an NL2SQL model without involving humans. For example, some work [6, 46, 91] uses an external model trained on a dataset of human attention to adjust the model attention, while some works may use a statistical [54] approach. Nevertheless, aligning attention automatically has limitations such as requiring the design of a model-specific alignment mechanism and the preparation of a dataset of human attention, which can not be generalized efficiently.

Our study of model attention provides a supportive theory for designing interactive approaches for NL2SQL models. In fact, existing interactive mechanisms can also be viewed as the implementation of attention alignment between the user and the model. For instance, MISP [88] asks clarification questions to users when uncertain about a generated token. This QA procedure aims to force the model's attention to align with the user's attention when the model's attention is very likely to be deviated. Our study also highlights the possibility of designing an attention-based error-handling mechanism. For example, the interactive approach can enable users to comprehend and directly manipulate the model's attention. Various approaches may employ different mechanisms at different layers of attention. But as long as humans are included in the loop, the model has an opportunity to align its attention with the human's attention through the model's explanation and user feedback.

6.2.3 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 [51]) and the NL explanations in step-by-step execution of the generated queries (in DIY [52]), 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.

6.2.4 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 error 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 mixed-initiative guides to facilitate error discovery and repair. Meanwhile, as discussed in Section 5.5, 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.

7 LIMITATIONS AND FUTURE WORK

The current study has several limitations. First, the total number is unbalanced for each error type (as shown in Section 3.5.1), which may cause bias in the study of error-handling mechanisms in Section 5. 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 have to exclude these error types in our analysis. The problem could be addressed by conducting larger-scale user studies with more participants and erroneous query data.

Second, despite that we reproduced four representative NL2SQL models based on the model architecture, it is hard to cover all due to the lack of open-source implementation or the engineering challenges in adapting them to our analysis pipeline. In addition, all the models used in our study are "black-box" models that do not provide much transparency into the process, which limits the selection of error-handling mechanisms. Interactive models [19, 73], 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.

Third, in the cause analysis study, we only explored one of the possible causes — attention misalignment, a more comprehensive analysis could be conducted to build up the factors that contribute to the model's performance. Additionally, we only tested one model in our study due to the computational resource and time limit. In future work, it could be valuable to explore recently emerged large open-sourced foundational models such as LlaMa-2 [78].

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.

8 CONCLUSION

In this paper, we i) presented an empirical study to understand the error types in the SQL query generated by NL2SQL models; ii) explored a possible cause of model errors by measuring the alignment of model-user attention on the NL query, and iii) conducted a controlled user experiment with 26 participants to measure the effectiveness and efficiency of representative NL2SQL error handling mechanisms. The error taxonomy summarized 48 types of

errors and revealed their distribution characteristics. Our study also demonstrated a strong correlation between the cause of model errors and the misalignment of attention between humans and models. 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, NSF grant CCF-2211428, and NSF grant ITE-2333736. 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* (Vancouver BC, Canada) *(CHIIR '20)*. Association for Computing Machinery, New York, NY, USA, 33–42. https://doi.org/10.1145/3343413.3377968

[2] James Allen, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift, and William Taysom. 2007. PLOW: A Collaborative Task Learning Agent. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2* (Vancouver, British Columbia, Canada) *(AAAI'07)*. AAAI Press, 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] Aakash Bansal, Bonita Sharif, and Collin McMillan. 2023. Towards Modeling Human Attention from Eye Movements for Neural Source Code Summarization. *Proc. ACM Hum.-Comput. Interact.* 7, ETRA, Article 167 (may 2023), 19 pages. https://doi.org/10.1145/3591136

[7] 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.

[8] 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.

[9] 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).

[10] 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).

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

[12] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What Does BERT Look at? An Analysis of BERT's Attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Florence, Italy, 276–286. https://doi.org/10.18653/v1/W19-4828

[13] 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.

[14] Misha Denil, Alban Demiraj, and Nando de Freitas. 2014. Extraction of Salient Sentences from Labelled Documents. *CoRR* abs/1412.6815 (2014). arXiv:1412.6815 http://arxiv.org/abs/1412.6815

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. ACL, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[17] 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.

[18] 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).

[19] 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*.

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

[21] Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. 2023. CatSQL: Towards Real World Natural Language to SQL Applications. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1534–1547.

[22] Andrea Galassi, Marco Lippi, and Paolo Torroni. 2021. Attention in Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems* 32, 10 (oct 2021), 4291–4308. https://doi.org/10.1109/tnnls.2020.3019893

[23] 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).

[24] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. arXiv:2308.15363 [cs.DB]

[25] 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.

[26] 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.

[27] 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.

[28] 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).

[29] 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.

[30] 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).

[31] 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).

[32] Siteng Huang, Min Zhang, Yachen Kang, and Donglin Wang. 2020. Attributes-Guided and Pure-Visual Attention Alignment for Few-Shot Recognition. *ArXiv* abs/2009.04724 (2020). https://api.semanticscholar.org/CorpusID:221586045

[33] 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).

[34] 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.

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

[36] 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).

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

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

[39] 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.

[40] 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.

[41] 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

[42] Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. Understanding Neural Networks through Representation Erasure. *CoRR* abs/1612.08220 (2016). arXiv:1612.08220 http://arxiv.org/abs/1612.08220

[43] 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

[44] 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).

[45] 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

[46] Xintong Li, Guanlin Li, Lemao Liu, Max Meng, and Shuming Shi. 2019. On the Word Alignment from Neural Machine Translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 1293–1303. https://doi.org/10.18653/v1/P19-1124

[47] 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).

[48] Shusen Liu, Zhimin Li, Tao Li, Vivek Srikumar, Valerio Pascucci, and Peer-Timo Bremer. 2019. NLIZE: A Perturbation-Driven Visual Interrogation Tool for Analyzing and Interpreting Natural Language Inference Models. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (jan 2019), 651–660. https://doi.org/10.1109/TVCG.2018.2865230

[49] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 4768–4777.

[50] 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

[51] 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.

[52] 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.

[53] 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 *Proceedings of the 28th International Conference on Intelligent User Interfaces* (Sydney, NSW, Australia) *(IUI '23)*. Association for Computing Machinery, New York, NY, USA, 633–649. https://doi.org/10.1145/3581641.3584067

[54] Franz Josef Och and Hermann Ney. 2003. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics* 29, 1 (03 2003), 19–51. https://doi.org/10.1162/089120103321337421 arXiv:https://direct.mit.edu/coli/article-pdf/29/1/19/1797914/089120103321337421.pdf

[55] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[56] 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.

[57] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs.CL]

[58] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *arXiv preprint arXiv:2304.11015* (2023).

[59] Marco Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Association for Computational Linguistics, San Diego, California, 97–101. https://doi.org/10.18653/v1/N16-3020

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

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

[62] 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.

[63] 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).

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

[65] 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

[66] 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.

[67] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning Important Features through Propagating Activation Differences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 3145–3153.

[68] 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.

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

[70] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. arXiv:1312.6034 [cs.CV]

[71] 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.

[72] 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.

[73] 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

[74] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. arXiv:1703.01365 [cs.LG]

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

[76] 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)*.

[77] Yuan Tian, Zheng Zhang, Zheng Ning, Toby Li, Jonathan K. Kummerfeld, and Tianyi Zhang. 2023. Interactive Text-to-SQL Generation via Editable Step-by-Step Explanations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 16149–16166. https://doi.org/10.18653/v1/2023.emnlp-main.1004

[78] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[79] 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.

[80] Shikhar Vashishth, Shyam Upadhyay, Gaurav Singh Tomar, and Manaal Faruqui. 2019. Attention Interpretability Across NLP Tasks. *CoRR* abs/1909.11218 (2019). arXiv:1909.11218 http://arxiv.org/abs/1909.11218

[81] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[82] 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).

[83] Xieyang Wang, Mengyi Liu, Jianqiu Xu, and Hua Lu. 2023. NALMO: Transforming Queries in Natural Language for Moving Objects Databases. *GeoInformatica* (2023), 1–34.

[84] 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.

[85] 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

[86] Zhiyong Wu, Yun Chen, Ben Kao, and Qun Liu. 2020. Perturbed Masking: Parameter-free Probing for Analyzing and Interpreting BERT. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 4166–4176. https://doi.org/10.18653/v1/2020.acl-main.383

[87] 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.

[88] 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

[89] 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).

[90] 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

[91] Jingyi Zhang and Josef van Genabith. 2021. A Bidirectional Transformer Based Alignment Model for Unsupervised Word Alignment. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 283–292. https://doi.org/10.18653/v1/2021.acl-long.24

[92] 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.

[93] 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.

[94] 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

[95] 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

[96] 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).

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

[98] 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.

[99] 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.