FISQL: Enhancing Text-to-SQL Systems with Rich Interactive Feedback

Rakesh R. Menon*
UNC Chapel Hill
rrmenon@cs.unc.edu

Kun Qian
Adobe Inc.
kunq@adobe.com

Liqun Chen
Adobe Inc.
lchen@adobe.com

Ishika Joshi
Adobe Inc.
ijoshi@adobe.com

Daniel Pandyan*
Texas A&M University
danielpandyan@gmail.com

Jordyn Harrison*
Towson University
jordyn@omnisynkai.com

Shashank Srivastava
UNC Chapel Hill
ssrivastava@cs.unc.edu

Yunyao Li
Adobe Inc.
yunyaol@adobe.com

ABSTRACT

Relational databases are foundational in managing and storing business-critical data across industries. However, accessing this data often requires specialized SQL knowledge, creating a barrier for non-technical users. To address this, research and industry have focused on developing text-2-SQL or natural language-to-SQL systems, allowing users to retrieve data with conversational inputs. While promising, these systems are prone to errors caused by ambiguities in user queries—especially with the unique vocabularies of closed-domain enterprise customers or vague phrasing from non-technical users. Such ambiguities, whether from domain-specific jargon or insufficient detail, make accurately interpreting user intent challenging.

Current NL2SQL systems often lack robust mechanisms for users to provide corrective input when their queries are misunderstood, resulting in a frustrating interaction. To bridge this gap, We present FISQL (Feedback-Infused SQL generation tool), an interactive framework designed to refine SQL query generation through user-provided natural language feedback and visual highlights. FISQL empowers users to iteratively correct, refine, or enhance SQL queries by offering intuitive ways to express feedback, which is then incorporated into the model's outputs.

FISQL dynamically updates SQL queries to align with user feedback, ensuring that subsequent generations are consistent with user expectations. Our evaluation demonstrates that FISQL significantly outperforms traditional query-rewrite mechanisms, achieving a 2× improvement in query correction accuracy on the SPIDER benchmark dataset.

1 INTRODUCTION

Developing natural language interfaces to databases (NLIDBs) has been a longstanding goal in the fields of natural language processing, human-computer interaction, and database research community[3, 9]. The allure of such systems lies in their ability to democratize access to information in databases by replacing knowledge requirements for database query languages with intuitive natural language dialogue.

Recent advancements have focused on text-to-SQL or natural language-to-SQL systems(In the rest of the paper, we use the term NL2SQL interchangeably to refer to both text-to-SQL and natural language-to-SQL) [12, 21, 22, 24], which enable users to query databases using conversational inputs. The emergence of large language models (LLMs)[1, 4, 15] in recent times has further revolutionized the field by reducing training requirements. With LLMs, few-shot in-context demonstrations are sufficient to provide supervision for models to excel on public NL2SQL benchmarks [7, 13, 20]. Additionally, the use of general-purpose LLMs, such as GPT-3.5, offers key benefits for enterprise production environments: no dedicated model training is required, and infrastructure maintenance can be minimized, significantly lowering operational overhead. While such systems have demonstrated strong performance on public datasets [26, 27], they face significant challenges in real-world, industry-specific use cases, where domain constraints and user behavior differ substantially from the public benchmarks.

One major challenge in enterprise use cases is the prevalence of closed-domain vocabularies and vague phrasing used by non-technical users. Unlike public NL2SQL benchmarks like SPIDER [26], which contain tables and columns with common-sense semantics, enterprise databases often include highly specialized schemas and domain-specific terminology. For instance, terms that have straightforward meanings in open domains may have entirely different or overloaded interpretations in closed domains. Similarly, non-technical users in enterprise environments frequently submit queries with incomplete or ambiguous phrasing, which further complicates SQL generation. These issues make it difficult for general-purpose LLMs, such as GPT-3.5, to generate accurate SQL queries for enterprise applications, even when leveraging Retrieval-Augmented Generation (RAG)-based [8] few-shot prompting techniques.

To empirically highlight the inherent difficulty of closed-domain use cases, we conducted experiments comparing NL2SQL performance of GPT-3.5-turbo (1106 subversion) model from OpenAI on SPIDER and an internal dataset derived from our Adobe Experience Platform (AEP) question traffic. We used simple, generic instructions in a zero-shot setting for GPT-3.5-turbo to perform NL2SQL tasks on both datasets. As shown in Figure 1, we provided general SQL generation instructions along with full schema definitions, but did not include relevant NL2SQL examples for in-context learning.

Figure 2 presents the accuracy results for both scenarios. On the SPIDER dataset, the LLM achieved an accuracy of 68.6%, while the accuracy dropped to 24% on the AEP dataset. The SPIDER

*Work done when interning at Adobe

You are a Nl2SQL agent designed to write an accurate SQL query given a question in natural language.

Please follow these instruction carefully:
- Given an input question, create a syntactically correct SQL query to run, and return the query.
- Never generate or include any DML (e.g., DELETE, INSERT, UPDATE) or DDL (e.g., CREATE, DROP, ALTER) SQL commands.
- You can order the results by a relevant column to return the most interesting examples in the database.
- Never query for all the columns from a specific table, only ask for the relevant columns given the question.
- DO NOT generate any other text apart from a SQL query.

The SQL database consists of several tables. Here is an overview of the tables and their columns: **{schema}**

Consider these rules to add LIMIT to the generated SQL Query: - Never add 'LIMIT' to queries which use aggregate functions like COUNT, AVG, etc. to return final output as an aggregate result.

Here is the question you need to answer:
Question: **{query}**
Query:

Figure 1: Zero-shot prompt skeleton used for AEP and SPIDER NL2SQL comparison.

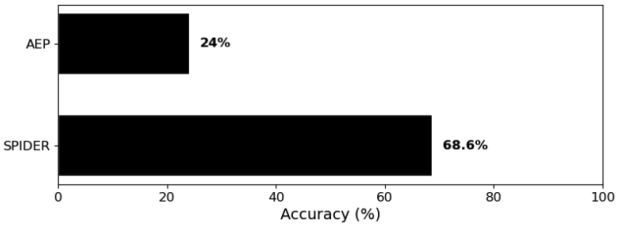

Figure 2: Accuracy comparison between SPIDER and AEP (Ours) datasets using zero-shot LLM prompts.

dataset features more common-sense schemas that are less ambiguous, and it is likely that the training data for SPIDER has been encountered or exposed to GPT models. As a result, general-purpose LLMs can generate correct SQL queries with relative ease, even without few-shot NL2SQL demonstrations. In contrast, the AEP case represents a more challenging, closed-domain scenario, where general-purpose LLMs like GPT-3.5-turbo often require additional clarifications and instructions to generate accurate SQL queries. For instance, consider the following AEP question

*which destinations is the 'ABC' segment **activated** to?.*

Here, the term '***activated***' is ambiguous. It could refer to a segment being in an active state, or, in the context of destinations, it could imply that the segment is connected to certain destinations. This connection can be determined by verifying that the inner joins between the segment table and the destination tables yield non-empty results. In fact, for this type of query, even with specific instructions and few-shot demonstrations, LLMs often struggle to generate semantically correct SQLs. This issue becomes more pronounced when customers rephrase the query, as the variations in phrasing can still lead to confusion and errors in interpretation by the LLMs.

To deal with these challenges, recent work has focused on the development of interactive systems that elicit feedback from users to guide SQL generations from NL2SQL systems. The format of the feedback elicited varies widely among different approaches, ranging from multiple choice questions [25] to natural language responses [5, 6]. However, in evaluations, these systems make a critical assumption of users possessing complete knowledge about the schema, and the ground-truth SQL command. In the real-world, there exists limited mechanisms for interaction due to the chat interface of an NLIDB and users that interact with an NLIDB often lack knowledge about the ground-truth SQL query and the underlying schema.

In this work, we introduce FISQL (pronounced *"physical"*) (**F**eedback-**I**nfused **SQL**), an interactive tool for *human-in-the-loop* NL2SQL correction with natural language as the main mode of interaction between the user and the system. Our tool features an intuitive interface, built on top of the Adobe Experience Platform [2], that leverages a custom-designed NL2SQL system to help customers access their information on a marketing analytics platform. Users on the platform can further provide feedback on the generated SQL in the form of natural language text to articulate any corrections or adjustments to the SQL query. Our tool incorporates this feedback in the context of the current chat to perform updates to the SQL queries, ensuring they match user intentions. Empirically, we validate the effectiveness of our tool by annotating errors in SQL generations made by an LLM-based NL2SQL model[1] on an in-house dataset featuring queries for the Adobe Experience Platform and SPIDER [26], and providing natural language feedback.[2]

For examples where natural language feedback can be provided by users, we observe that FISQL is able to incorporate the feedback effectively and generate correct SQL queries in 40-70% of the cases over multiple interaction rounds across datasets. Compared to a query-rewrite mechanism, that reformulates the user query to incorporate information available in the feedback, we observe that FISQL is able to correct nearly 2× the number of queries using the LLM-based approach. This highlights the efficacy of our tool in utilizing rich natural language feedback from users to guide NL2SQL systems.

2 RELATED WORK

Natural Language to SQL. Generating SQL queries from natural language questions provided by users has been studied extensively in the past few decades [18, 19]. However, these systems introduced in these works made strong assumptions on the SQL query structure limiting generations to specific SQL operations or required questions to follow specific syntax rules. With advancements in deep learning, systems that utilize sequence-to-sequence models trained on large-scale datasets, such as SPIDER [26] or WikiSQL [27], has become commonplace [21, 22, 24]. However, the limited availability of annotated datasets, particularly in industry settings, limited their widespread adoption. In recent years, the emergence of LLMs has ameliorated data requirements and reduced barriers for adoption of NLP systems in production. LLMs, prompted with a few in-context demonstrations, achieve state-of-the-art performance on public benchmarks [7, 13, 20].

[1] We use `gpt-3.5-turbo-1106` model from OpenAI.
[2] We are going through an internal process to open-source the feedback annotations for the SPIDER dataset to facilitate further research.

Nonetheless, these models frequently make errors caused by question misunderstanding, or hallucinations and struggle to answer complex questions in a single attempt. Feedback is therefore necessary to improve system performance. In this work, we focus on systems that incorporate feedback into their pipeline to improve SQL generations and match user expectations.

Utilizing feedback to improve NL2SQL. Recent years have witnessed an increased traction towards techniques that interact with users and incorporate feedback to improve NL2SQL generations. The ways in which systems interact and elicit feedback from users can take multiple forms: through feedback buttons, like as thumbs up-thumbs down [10], selection of alternative sub-queries [16], or multiple-choice questions [25]. However, the constrained feedback format leads to limited opportunities to improve model generations.

More recent works propose to utilize natural language feedback from users to guide SQL generations from NL2SQL models [6, 23]. However, all of these approaches assume that the user, who provides feedback, has complete knowledge of the ground-truth [SPLASH, 5] and perhaps more inconspicuously, an assumption that users are experts at SQL and hence can provide directed feedback. In this work, we address methods that can handle feedback of this form effectively through different prompt-based approaches.

3 FISQL

In this section, we describe our tool FISQL, short for **F**eedback-**I**nfused **SQL** generation tool. Our tool is designed for incorporating user feedback provided in the form of natural language text to align SQL generations with user intentions. In the next few subsections, we provide an overview of our tool and the techniques employed in the backend to incorporate user feedback.

3.1 Interface

We build our tool on top of the Adobe Experience Platform (AEP) Assistant [2]. AEP is a powerful, open, and flexible system designed to build and manage complete solutions that drive experiences of marketing organizations. The platform is built on RESTful APIs supporting easy integration of enterprise solutions using familiar tools. It helps brands reimagine their marketing strategy, accelerate transformation efforts, and deliver consistent personalization at scale. By logging customer interactions and customer experience challenges, AEP empowers organizations to exceed customer expectations and deliver fully-personalized experiences drawn from a 360-degree view of the customer and their individual preferences.

In Figure 3, we depict the entry point into the platform and the associated AI assistant that can be accessed to interact with organizational data. As shown here, users of the platform can integrate data across sources to fuel personalized campaigns and customer journeys using Real-Time Customer Data Platform (Real-Time CDP) and Adobe Journey Optimizer, enhancing customer engagement and loyalty. A brand can also merge data with sources like Adobe Audience Manager and Adobe Campaign to enrich customer profiles, create cross-channel campaigns, and deliver AI-powered personalization. On the left side of Figure 3, users can access information uploaded into the Experience Platform. On the right side of Figure 3, the platform also provides access to an AI assistant[3] (accessible through button near Figure 3A) that can aid with more directed information access with respect to available organizational data and provide operational insights over the available data.

[3]hitherto referred to as "Assistant" [14].

3.2 Generating Operational Insights Using Assistant in FISQL

Given the recent success of LLMs in achieving state-of-the-art performance on text-to-SQL generation benchmarks [7], the Assistant in our tool employs `gpt-3.5-turbo` as the base NL2SQL model with an in-house developed prompt. More concretely, the NL2SQL model utilizes a retrieval-augmented generation [RAG, 11] approach to adaptively draw user query-relevant SQL demonstrations to aid the model in generation of the SQL command. In Figure 3, users can ask queries in the dialogue box (shown in Figure 3B). In response, the Assistant in our tool, generates a SQL query, executes the command against internal databases, and returns four text-based outputs: (a) the execution result, (b) a reformulation of the user query – indicating the Assistant's understanding of the question, (c) a natural language explanations of the steps undertaken to answer the user query, and finally (d) the SQL output itself – accessible via a 'Show Source' button (shown in Figure 3C).

3.3 Feedback Incorporation

To provide a lightweight extension of the existing NL2SQL pipeline that employs LLMs, our feedback strategy utilizes prompting techniques to guide the SQL generations, which we describe in more detail. This occurs through a two-step prompting procedure. First, we utilize a prompt-based approach to identify the type of update requested by the users and retrieve examples that demonstrate how to perform necessary updates. Second, we prompt the NL2SQL model with the examples from the previous step and the feedback provided by the user to regenerate the SQL query.

Feedback-type Identification. To perform precise revisions to SQL generations based on user feedback, LLMs require demonstrations that illustrate how to perform necessary updates to SQL queries. For example, consider the chat between a user and Assistant on FISQL, shown in Figure 4. Given an incorrect response from the Assistant due to query misunderstanding, a user can prompt the Assistant with feedback of the form *'we are in 2024'* to indicate the incorrect segment of the SQL generation in the Assistant's response. In an ideal interactive system, a feedback of the form *we are in 2024*' should exclusively edit the year `2023` operation to `2024`.

To instruct LLMs with pertinent demonstrations, we first categorize the kind of user feedback into three categories based on the kind of operation suggested by the feedback. Here, we divide user feedback into three types: (i) **Add**, (ii) **Remove**, and (iii) **Edit**. An example of each type of feedback is shown in Table 1. Here, the **Add**/**Remove** type correspond to feedback that suggest the addition/removal of SQL operations in a query. **Edit** refers to feedback that updates arguments of SQL operations. To identify the feedback type, we prompt a `gpt-3.5-turbo` model with a few demonstrations that illustrate how to perform the feedback categorization. Once the operation type is identified, we retrieve a fixed set of examples that illustrate how to revise SQL queries based on the predicted feedback type. We then append these examples to the in-context demonstrations for the NL2SQL model. Note that these demonstrations are in addition to the examples retrieved by the RAG pipeline. An example of the demonstration for the Edit operation has been shown in Figure 5.

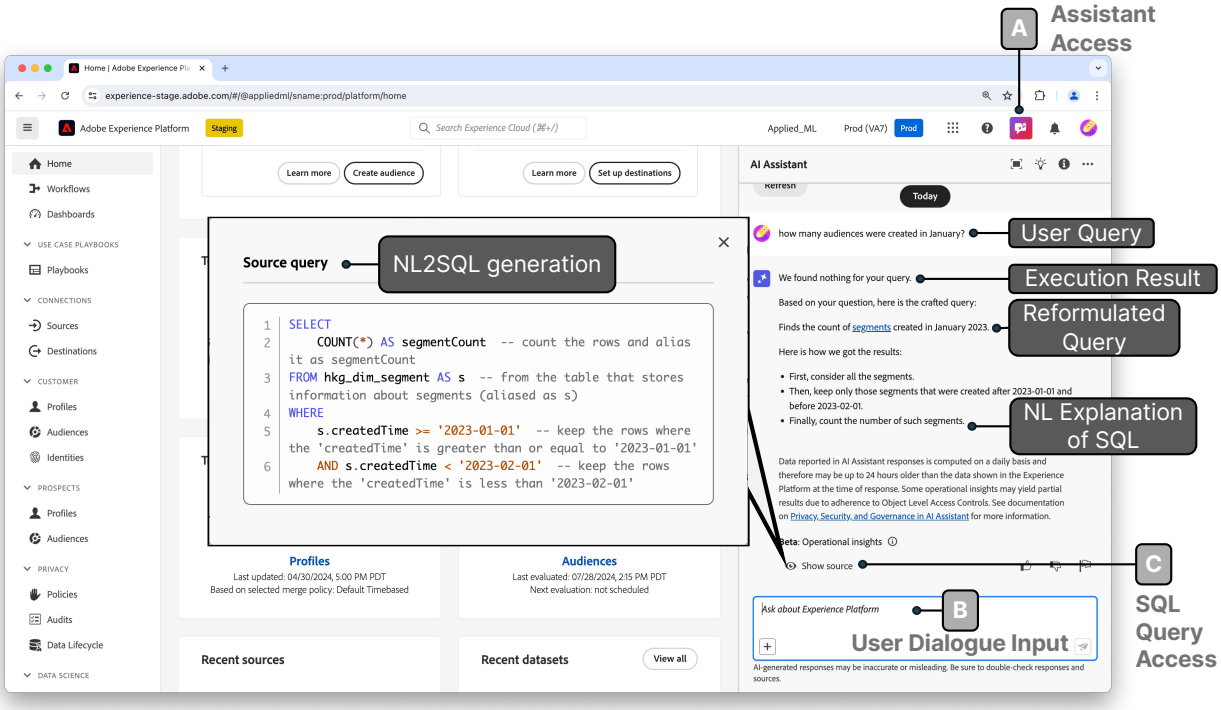

Figure 3: Adobe Experience Platform (AEP) interface provides various options for a brand to interact with its customer data and offer data-driven personalized insights thus transforming their marketing strategies. Users of the platform can (A) access the onboard AI assistant to (B) ask questions about various marketing insights. In addition to the execution results and step-by-step explanations of procedures employed by the AI Assistant, users also (C) have access to in-line commented SQL queries generated by in-house NL2SQL models.

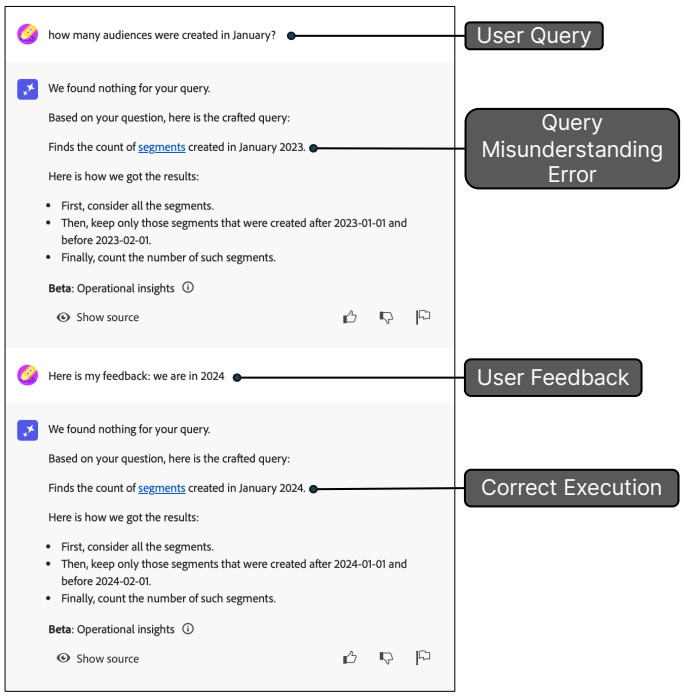

Figure 4: Using our feedback mechanism, users can converse with the AI Assistant on the AEP platform to correct ambiguities, underspecifications, or model misunderstanding errors.

Question: *How many audiences were created in January?*

Query: `SELECT COUNT(*) AS segmentCount FROM hkg_dim_segment WHERE createdTime >= '2023-01-01' and createdTime < '2023-02-01'`

The SQL query you have generated has received the following feedback: *we are in 2024*

Taking into account the feedback, please rewrite the SQL query.

Query: `SELECT COUNT(*) AS segmentCount FROM hkg_dim_segment WHERE createdTime >= '2024-01-01' and createdTime < '2024-02-01'`

Figure 5: An example that demonstrates how to perform an Edit-type operation based on the user feedback. Each feedback example is structured similarly when passed to the NL2SQL model.

Feedback Type	Feedback Text
Add	*order the names in ascending order.*
Remove	*do not give descriptions*
Edit	*increase the limit to 10*

Table 1: Examples illustrating different feedback types considered in this work. The feedback sentences correspond to feedback generated for the SPIDER dataset [26].

NL2SQL prompting. In regeneration of SQL queries based on user feedback, we minimally update the prompt for standard

{Instructions on how to perform the NL2SQL task.}

{In-context demonstrations from RAG + *Feedback demonstrations*}

Here is the question you need to answer:
Question: {question}
Query: *{initial_predicted_query}*
The SQL query you have generated has received the following feedback: {feedback}
Taking into account the feedback, please rewrite the SQL query.
Query:

Figure 6: Minimal illustration of the prompt to make the NL2SQL model account for user feedback while generating SQL command. Changes made to the NL2SQL prompt have been put in *italics*.

NL2SQL execution. This has been illustrated in Figure 6, where we pass in the SQL query generated in the previous turn along with the corresponding feedback provided by the user.

4 EVALUATION

To evaluate the efficacy of NL2SQL systems with corrective feedback, we perform evaluations over two datasets: (a) an in-house dataset comprising of Adobe Experience Platform queries where the Assistant fails to answer in one-shot, and (b) a dataset with annotated feedback for errors made by an Assistant on SPIDER [26]. While some recent works study the errors made by GPT-level models in an interactive setup [17], they do not open-source annotations for feedback provided by users. Furthermore, as discussed in Section **??**, interactive NL2SQL systems, that ingest natural language feedback, typically assume that users providing feedback have knowledge about the ground-truth SQL and ask them to suggest minimal edits that would correct the SQL queries to match the intention of the original question. However, this is an unrealistic assumption in the real-world, where AI assistants receive feedback from users who have varied levels of knowledge about SQL without clear access to the ground-truth SQL query. Further, companies may want to limit user access to underlying schema to protect intellectual property. To account for these factors, we perform a feedback collection for errors made by the NL2SQL model employed in Assistant. Next, we describe the experimental setup and discuss preliminary results using the curated dataset.

4.1 Experiment Setup

Here, we describe the process of feedback collection used for the public dataset, SPIDER, as well as in-house dataset used for evaluating the utility of the tool for the Assistant. Further, we provide information about the baselines and metrics used in our evaluation.

SPIDER Feedback Collection. In the SPIDER dataset, there are about 200 databases with 5-20 tables per database and 5-10 columns per table. We utilize all available schemas and introduce a new dataset based on errors made by a `gpt-3.5-turbo` model on the SPIDER validation set [26]. The feedback annotations are provided by authors of this work exclusively based on information available from the tool, i.e., question, the SQL query, natural language explanation of the SQL query, and the SQL evaluation

Question: Show the name and the release year of the song by the youngest singer.

SQL Query:
SELECT Name, Song_release_year
FROM singer
WHERE Age = (SELECT min(Age) FROM singer)

SQL Explanation:
- We selected the name and release year of the song from singer table
- We filtered the results to only include the singer with the minimum age.

Evaluation:

Name	Release Year
Tribal King	2016

Feedback:
Provide song name instead of singer name

Figure 7: An example of the information available during the feedback collection from users for SPIDER.

against the database when available. Figure 7 shows an example of the data available during the annotation procedure for SPIDER.

In our setup, `gpt-3.5-turbo` makes 243 errors out of the 1034 available datapoints in the validation set for SPIDER. Of these, we annotate feedback for 101 examples (~ 41% of the errors) using the available information for the user. We use the collected feedback in order to evaluate the efficacy of our tool in incorporating corrective information.

Experience Platform Dataset. Our primary evaluation, however, is based on queries made on the Adobe Experience Platform. This is an internally developed dataset, designed to capture and analyze instances in which users were unable to attain their desired outcomes when interacting with the Assistant. By examining these specific cases, we aim to gain insights into the challenges and limitations faced by real-world users. The in-house dataset (tagged Experience Platform in the results), contains 54 examples of queries that Assistant fails to respond to correctly in the one-shot given the user query.[4] The authors of this work undertake the task of providing feedback for these errors to match user intent.

Baselines and Metrics. For comparison, we evaluate an additional query reformulation baseline (**Query Rewrite**) – where given the initial user query and the feedback from the user, a paraphrasing model (instantiated using `gpt-3.5-turbo` with few-shot examples) resolves the feedback and generates a new query that captures information from both user inputs, Finally, for our metric, we measure the percentage of instances where providing feedback helped the model achieve the correct SQL execution result.

[4]We avoided semantic enrichment to maintain consistency with SPIDER settings.

Method	% Instances Corrected (Experience Platform)	% Instances Corrected (SPIDER)
Query Rewrite	35.85	16.83
FISQL (- Routing)	-	43.56
FISQL	**67.92**	**44.55**

Table 2: Percentage of instances corrected by using natural language feedback for errors made by Assistant.

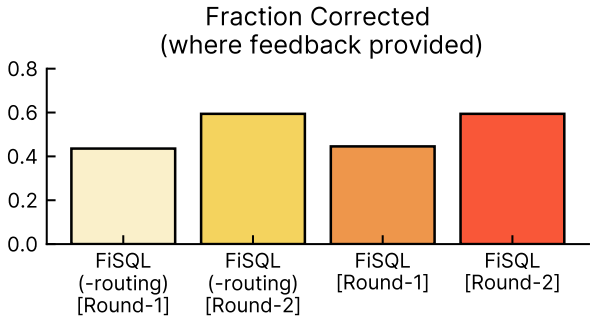

Figure 8: Improving performance of the NL2SQL models through additional feedback rounds using FISQL and FISQL (- Routing) on SPIDER errors.

4.2 Results

In Table 2, we summarize the results using the different feedback incorporation techniques across the Experience Platform and SPIDER datasets. We note that Query Rewrite performs significantly worse compared to FISQL on both datasets. Quantitatively, FISQL is able to correct nearly 2× the number of examples using user feedback, underscoring its enhanced efficiency.

To measure the value of routing, we explore an additional baseline by ablating the routing mechanism (FISQL (- Routing)) for the SPIDER dataset errors. Here, we observe that routing to identify feedback operation types offers an advantage over naively incorporating feedback. On closer observation, we notice the improvement emerges as a result of the precise revisions performed by FISQL as opposed to the ablated version without routing.

Finally, we observe that none of the approaches are close to achieving 100% error correction. While performing an error analysis, we observed that this is caused by the examples either: (a) having SQL queries with multiple errors and hence needing multiple feedback rounds, (b) inability of the approaches to interpret user feedback and make edits to the SQL query, and (c) user feedback being misaligned with the correction required for the SQL query (note that users do not have access to the ground-truth SQL query or its execution result in this settings).

FISQL improves with further feedback. We incorporate an additional feedback round to correct the errors made by FISQL and FISQL (-routing) approaches on the Assistant's errors in the SPIDER dataset. Specifically, we conduct one additional feedback round to correct examples, through which we were able to improve performance of the model by nearly 15% for each of the two approaches (illustrated in Figure 8). After two feedback rounds, we observe that the FISQL (- Routing) approach has corrected the same errors as FISQL. This further demonstrates how our tool can be utilized by users to correct multiple errors sequentially.

Question: *How many audiences were created in January?*
Query: `SELECT COUNT(*) AS segmentCount FROM hkg_dim_segment WHERE createdTime >= '2023-01-01' and createdTime < '2023-02-01'`
User Feedback: change to 2024.

Figure 9: An example that demonstrates how to users can highlight portions of the SQL query to ground suggestions for SQL improvement. The highlight is shown using underline in the text.

Additional feedback through highlights enhances performance of FISQL.. As noted in the error analysis, some of the remaining errors after applying the approach in FISQL arise from the models' inability to effectively ground user feedback to the SQL query, thereby hindering their capacity to provide appropriate corrections. To alleviate this issue, we allow users ground their language feedback by highlighting segments of the SQL query (or its natural language explanation). An example is shown in Figure 9 where the user can highlight the segment corresponding to the `WHERE` clause to ground the suggestion *'change to 2024'*.

Method	% Instances Corrected (Experience Platform)	% Instances Corrected (SPIDER)
FISQL	67.92	**44.55**
FISQL (+ Highlighting)	**69.81**	**44.55**

Table 3: Percentage of instances corrected with highlights and natural language feedback for errors made by Assistant.

In Table 3, we demonstrate the empirical value of highlights on the Experience Platform and SPIDER datasets. Here, we observe that the additional use of highlights results in a performance improvement on the Experience Platform while maintaing the same error reduction rate for the SPIDER dataset. Hence, through addition of highlighting, we can drive enhance user experience and drive more engagement with interactive NL2SQL systems.

5 CONCLUDING REMARKS

We presented FISQL, a tool designed to foster interactive conversations between users and NL2SQL systems. This tool has been implemented as been tested as a part our product and we aim to continually improve the feedback integration capabilities of our tool as usage evolves. Currently, our empirical studies evaluate the performance of our tool for scenarios where users perform error corrections to complex SQL queries. There are multiple avenues of future work that can be studied with our tool. First, this tool could be adapted to allow users to build up complex SQL queries by asking simple questions first. Second, our routing mechanism can be enhanced with dynamic example selection based on query structure and feedback. We plan to

evaluate these extensions through a user study, which will be reported in subsequent work.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Anjul Bhambhri. 2024. New ai integrations in adobe experience platform. https://business.adobe.com/blog/the-latest/new-ai-assistant-in-adobe-experience-platform

[3] E. F. Codd. 1974. Seven Steps to Rendezvous with the Casual User. In *IFIP Working Conference Data Base Management.* https://api.semanticscholar.org/CorpusID:28690513

[4] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[5] Ahmed Elgohary, Saghar Hosseini, and Ahmed Hassan Awadallah. 2020. Speak to your Parser: Interactive Text-to-SQL with Natural Language Feedback. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 2065–2077. https://doi.org/10.18653/v1/2020.acl-main.187

[6] Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo Ramos, and Ahmed Hassan Awadallah. 2021. NL-EDIT: Correcting Semantic Parse Errors through Natural Language Interaction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 5599–5610. https://doi.org/10.18653/v1/2021.naacl-main.444

[7] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proceedings of the VLDB Endowment* 17, 5 (2024), 1132–1145.

[8] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:cs.CL/2312.10997 https://arxiv.org/abs/2312.10997

[9] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. 1977. Developing a natural language interface to complex data. In *TODS*. https://api.semanticscholar.org/CorpusID:15391397

[10] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, Vancouver, Canada, 963–973. https://doi.org/10.18653/v1/P17-1089

[11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[12] Yunyao Li, Dragomir R. Radev, and Davood Rafiei. 2023. *Natural Language Interfaces to Databases.* Springer Nature.

[13] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability. *arXiv preprint arXiv:2303.13547* (2023).

[14] Akash Maharaj, Kun Qian, Uttaran Bhattacharya, Sally Fang, Horia Galatanu, Manas Garg, Rachel Hanessian, Nishant Kapoor, Ken Russell, Shivakumar Vaithyanathan, and Yunyao Li. 2024. Evaluation and Continual Improvement for an Enterprise AI Assistant. In *Proceedings of the Fifth Workshop on Data Science with Human-in-the-Loop (DaSH 2024)*, Eduard Dragut, Yunyao Li, Lucian Popa, Slobodan Vucetic, and Shashank Srivastava (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 17–24. https://doi.org/10.18653/v1/2024.dash-1.3

[15] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).

[16] Arpit Narechania, Adam Fourney, Bongshin Lee, and Gonzalo Ramos. 2021. DIY: Assessing the Correctness of Natural Language to SQL Systems. In *Proceedings of the 26th International Conference on Intelligent User Interfaces (IUI '21).* Association for Computing Machinery, New York, NY, USA, 597–607. https://doi.org/10.1145/3397481.3450667

[17] Zheng Ning, Yuan Tian, Zheng Zhang, Tianyi Zhang, and Toby Jia-Jun Li. 2024. Insights into Natural Language Database Query Errors: From Attention Misalignment to User Handling Strategies. *ACM Trans. Interact. Intell. Syst.* (mar 2024). https://doi.org/10.1145/3650114 Just Accepted.

[18] Hoifung Poon. 2013. Grounded Unsupervised Semantic Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Hinrich Schuetze, Pascale Fung, and Massimo Poesio (Eds.). Association for Computational Linguistics, Sofia, Bulgaria, 933–943. https://aclanthology.org/P13-1092

[19] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces (IUI '03).* Association for Computing Machinery, New York, NY, USA, 149–157. https://doi.org/10.1145/604045.604070

[20] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498* (2022).

[21] Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 311–324. https://doi.org/10.18653/v1/2021.naacl-main.29

[22] 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*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779

[23] 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

[24] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[25] 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)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 5447–5458. https://doi.org/10.18653/v1/D19-1547

[26] 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*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. https://doi.org/10.18653/v1/D18-1425

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