# Utilizing Past User Feedback for More Accurate Text-to-SQL

Matthias Urban*
Technical University of Darmstadt

Jialin Ding
Amazon Web Services

David Kernert
STACKIT

Kapil Vaidya
Parallel Web Systems

Tim Kraska
Amazon Web Services

## Abstract

In the classical problem formulation of Text-to-SQL in academia, each question is translated independently of the others into SQL. This differs from the setting in practice, where questions enter the Text-to-SQL system in sequence. Thus, for all but the first few questions, a translation history is available that contains past questions and how they were translated by the system. So far, it has not been sufficiently explored how Text-to-SQL systems can make use of the translation history to improve future translations. Another crucial difference from the academic setting is that in practice, users generally have a conversation with the Text-to-SQL system. More concretely, if the initial translation contains a mistake, the user can follow up with feedback messages to allow the system to fix the mistake. In this case, it might be helpful to remember these user feedback messages to avoid repeating past mistakes. Thus, in this paper, we explore how a history of such past conversations between users and the Text-to-SQL system can be used to make future Text-to-SQL translations more accurate. We explore several approaches for extracting relevant experiences and insights from this conversation history and show in an evaluation that utilizing them can improve translation accuracy by up to 14.9%.

## Keywords

Text-to-SQL, Natural Language Interface for Databases, NL2SQL

**ACM Reference Format:**
Matthias Urban, Jialin Ding, David Kernert, Kapil Vaidya, and Tim Kraska. 2025. Utilizing Past User Feedback for More Accurate Text-to-SQL. In *Workshop on Human-In-the-Loop Data Analytics (HILDA' 25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3736733.3736739

## 1 Introduction

Text-to-SQL systems greatly reduce the barrier of entry for non-expert users of databases, as they allow users to interact with the database in natural language. For more experienced users, they are a useful tool to bootstrap the formulation of complex SQL queries. With the rise of Large Language Models (LLMs), Text-to-SQL systems have experienced a significant boost in accuracy and reliability, leading to a surge of interest from both industry [e.g., 8, 17, 14, 7] and academia [e.g., 19, 4, 10, 11, 16].

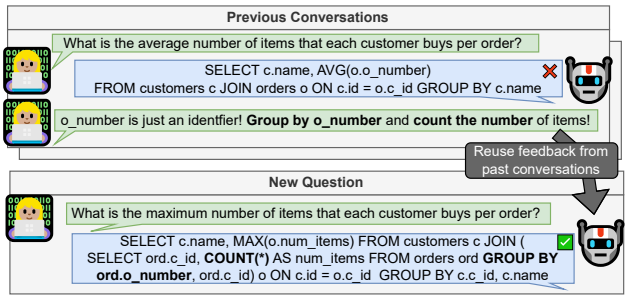


**Figure 1: Previous conversations of users with a Text-to-SQL system can be used to make future Text-to-SQL translations more accurate. (top) In a past conversation, the user corrects the Text-to-SQL system on correctly counting items in an order. (bottom) For similar future queries, we can use that explanation to avoid making the same mistake. As a result, the system can output the correct SQL immediately.**

**The difference between academia and practice.** However, most systems proposed by the literature follow the task definition of the two most popular Text-to-SQL benchmarks, Spider [23, 9] and Bird [12], which differ from the setting in practice. In the benchmarks' task definition, the system gets a natural language question as input and should output the SQL query that answers the question. To generate a SQL query grounded in the available data, the system has access to the database, i.e., the database schema and the stored data. Sometimes, for example, in the case of Bird, there is additional external knowledge available that most systems choose to utilize as well [e.g., 8, 19]. This setting differs from the setting in practice in two important ways. First of all, in practice Text-to-SQL systems are typically conversational [1], meaning that the user is able to interact with the system to correct or modify generated SQL statements using natural language feedback as shown in Figure 1 (top). Secondly, the system typically has access to more information than just the input question and the data. For instance, it might have access to past conversations of users with the Text-to-SQL system.

**Text-to-SQL systems are conversational.** Existing Text-to-SQL systems like Amazon Q Generative SQL [1] allow users to have a conversation with the Text-to-SQL system. In these conversations, users provide feedback messages to fix SQL queries or to modify them according to the user's preferences. Thus, they contain valuable information about user preferences and data that can help to translate future questions. For instance, in the example of Figure 1 (top), the user asks about the average number of items each customer bought per order. With only the schema as context, the

*Work done during an internship at Amazon Web Services.


*HILDA' 25, June 22–27, 2025, Berlin, Germany*

ACM ISBN 979-8-4007-1959-2/2025/06
https://doi.org/10.1145/3736733.3736739

question can be difficult to translate correctly. And indeed the Text-to-SQL system misunderstands the schema and makes a mistake. It confuses a column called `o_number` as the number of items per order, while it is, in fact, just an identifier. The user spots the mistake and provides feedback that allows the Text-to-SQL system to fix it. More concretely, we assume the following interaction model: each conversation of a user with the Text-to-SQL systems starts with an input question to which the system responds with a translation attempt. When the user is satisfied with this SQL statement, they can click an `accept`-button to mark it as correct. Otherwise, the user can follow up with feedback messages to fix mistakes or to modify the SQL statement according to their preferences (e.g., they might prefer some additional columns for context). The conversation continues until either a satisfying SQL statement is generated, in which case the user clicks the `accept` button, or the user gives up and leaves the conversation without clicking the `accept` button.

**Leveraging the conversation history.** The key idea of this paper is to utilize these past conversations (the conversation history) and, in particular, the user's feedback messages when we translate new questions. In the example above, we want to avoid making the same mistake as in the past. Instead, we aim to use the information provided by the user feedback that `o_number` does not store the number of items per order. This allows us to immediately output the correct SQL statement without any additional human feedback.

A straightforward way to exploit past conversations is to use the input question and the final output SQL statement of `accepted` past conversations as few-shot examples for new questions. As we show in our evaluation, this can already lead to some improvements, especially if the pool of past conversations contains many relevant conversations and the extracted few-shot examples are selected carefully. However, in this paper, we additionally utilize the users' feedback messages from past conversations for additional accuracy gains of over 5% on some datasets.

**Challenges.** So far, it has not been explored how the conversation history can be used most effectively to make new Text-to-SQL translations more accurate. One problem is that the number of conversations in the conversation history can be arbitrarily large. Moreover, each conversation can contain multiple messages from the user and the Text-to-SQL system. Hence, simply pre-pending the entire conversation history to the prompt of a new question in an in-context learning fashion quickly becomes infeasible. On the other hand, fine-tuning new models on the conversation history comes with large training overheads, as models would need to be retrained regularly when new conversations come in. Another problem is that when users formulate feedback messages, they always have a particular question or mistake regarding a SQL query in mind. Thus, they might not generalize to new questions. In fact, reusing them can even be detrimental to the accuracy. For instance, in Figure 1, the user provides the feedback *Group by o_number and count the number of items!* This feedback message was very helpful for the specific input question asking about the number of items per order. However, for new questions that do not ask about items per order and that do not need a group-by statement, this feedback message will be irrelevant or even misleading.

**Extract and retrieve insights from the conversation history.** To tackle the abovementioned challenges, we propose a two-staged

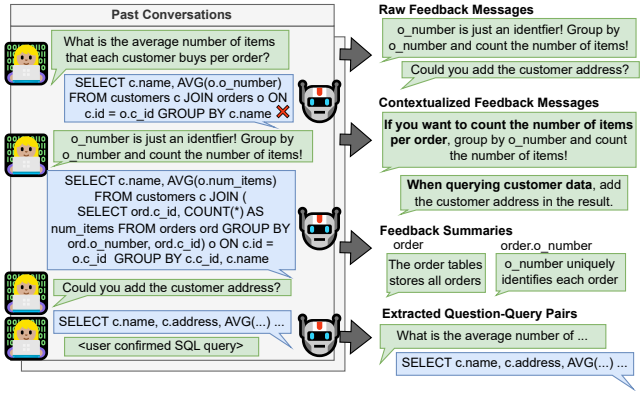


**Figure 2: Different approaches to extract the relevant insights from the conversation history. *Raw Feedback Messages* is a rule-based approach that simply extracts the user's feedback messages as-is. *Contextualized Feedback Messages* adds the context to the messages using an LLM. *Feedback Summaries* uses an LLM to summarize what can be learned from a single conversation about individual components of the schema, such as tables and columns. Finally, *Extracted Question-Query Pairs* simply extracts the input question and the final SQL statement and uses it as few-shot examples.**

procedure detailed in Section 2. The first stage is an offline pre-processing stage, where we extract the most relevant insights from the conversation history. These insights can then be used as additional metadata that is helpful for translating new questions, such as that `o_number` is an identifier and not the number of items per order. We experiment with several rule-based and LLM-powered insight extraction methods as explained in Section 2.1. In the second stage (online), we retrieve the relevant insights given a new question. Here, we are inspired by recent schema-linking approaches [19], which is the task of selecting relevant tables and columns for a given question. Those approaches use LLM-prompting to select the most relevant tables and columns. Similarly, we also select insights from past feedback using an LLM, as explained in Section 2.2.

**Contributions and outline.** In this paper, we present and compare several ideas for tackling both challenges of extracting and retrieving insights from a conversation history. We present an evaluation on public and internal benchmarks and show that using such insights can substantially increase translation accuracy. The remainder of the paper is structured as follows: in Section 2, we present our solutions for extracting insights from the conversation history (Section 2.1) and our solution for retrieving insights (Section 2.2). Afterwards, in Section 3, we show the benefits of utilizing the conversation history in an evaluation, present limitations and future work in Section 4, and conclude in Section 5.

## 2 Learning From the Conversation History

To learn from the conversation history, we first extract relevant insights from the conversation history offline. Then, when a new question enters the system, we retrieve relevant feedback items from a potentially large pool of such insights. Finally, we use those retrieved insights to translate the new question.

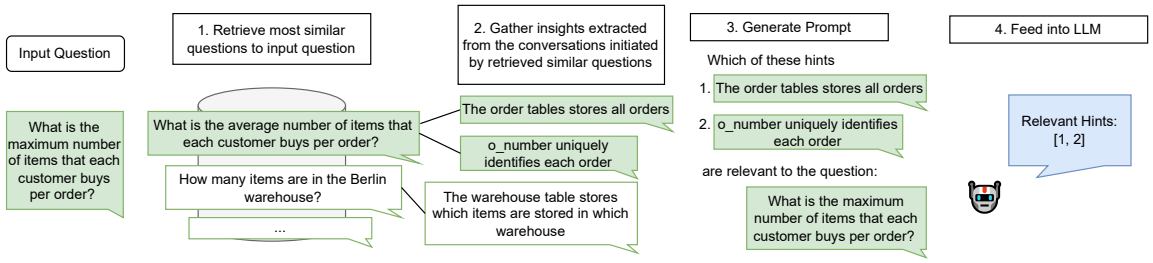


**Figure 3: Overview of how insight retrieval and selection from the conversation history works. Since the size of the conversation history can be arbitrarily large, we first need to narrow down the number of conversations. Hence, in the first step, we embed the input question using Amazon Titan Text Embeddings [2] and retrieve similar past questions. Next, we gather all the insights extracted from the conversation initiated by these retrieved questions. Finally, we put those feedback items into a prompt and instruct an LLM to select the most relevant insights to the current question.**

## 2.1 Extracting Insights from Conversations

The first task is to extract relevant insights from the conversation history. An insight can be anything from additional information about certain tables or columns, which can help understand the data better, to user preferences, such as the user's preference for additional context-providing columns in the output. For example, the user could prefer not to limit the output to just the name of the customer and the average number of items for the question in Figure 1. Instead, they might prefer to have some context columns, such as the customer ID and the headquarters location. To extract these insights from the conversation history, we propose four extraction methods that we later compare in our evaluation in Section 3. All approaches are summarized in Figure 2.

**Raw Feedback Messages (RF).** A simple approach to extracting insights from the conversation history is simply extracting the user's feedback messages. These were helpful in the previous conversations to fix the translation of a previous question and can thus also help translate new questions. However, as they have been formulated with a particular question in mind, they can be misleading for new questions, as mentioned before. Thus, it is especially important to have good feedback retrieval, which we explain in Section 2.2. See Figure 2 (top) for an example where the raw feedback message *'o_number is just an identifier. Group by o_number and count the number of items'* has been extracted from a previous conversation.

**Contextualized Feedback Messages (CF).** However, as illustrated by this example, raw feedback messages are potentially misleading for the translation of future questions (e.g., if they do not require grouping by `o_number`). To avoid potentially misleading insights that are only relevant to one particular question from the past we inject that question's *context* into the feedback item. In the example from Figure 2 we inject the context of counting items per order into the feedback messages to obtain *'If you want to count the number of items per order, group by o_number and count the number of items'*. To achieve this, we use an LLM that takes a whole conversation from the conversation history as input. Then, we prompt it to reformulate the individual feedback messages to include the context in the form of a condition that conveys when the feedback is relevant. We use a single prompt per conversation, which, in addition to the conversation and the instruction, also contains the database schema to let the LLM better understand the context it should generate. The output is one piece of contextualized feedback per user message.

**Feedback Summaries (FS).** One issue with contextualized feedback is that the resulting feedback is very specific. For instance, consider the example from Figure 2, where the system was confused by `o_number`, and the user helped to clear up the confusion with their feedback. The missing piece of information that caused the confusion was that `o_number` is an identifier and not the number of items per order. However, none of the generated contextualized feedback messages explicitly contain this information and instead only contain hints for very specific situations, such as for counting the number of items. Hence, to explicitly extract what can be learned from the conversation history, we also evaluate a method that summarizes the most important insights from past conversations using LLM prompting. In particular, we use a single prompt per conversation to output what can be learned about the user and the database in general, as well as different schema components, such as the available tables and columns (which are also part of the prompt). This allows us to attach tailored insights about columns and tables to the table and column metadata. As we will see, combining this approach with *Extracted Question-Query Pairs*, which we explain next, consistently achieves the best improvements across all data sets in our evaluation in Section 3.

**Extracted Question-Query Pairs (EQQ).** Finally, we can extract the input question and the final SQL statement from past conversations to obtain few-shot examples. This approach, despite its simplicity, turns out to be quite powerful. An evident reason for the effectiveness of this approach is that the final SQL statement results from all user feedback messages and thus summarizes the information of the entire conversation in a single statement. Moreover, LLMs are good at learning from task demonstrations [3].

Note that these approaches can be freely combined to obtain a larger pool of feedback. In fact, as we show in Section 3, combining EQQ with FS performs best across data sets.

**Discussion.** The previously presented four ways to extract insights cover the space of possible insight extraction methods well. CF and FS use Large Language Models for post-processing, RF and EQQ are rule-based. All of them differ on which information is included in the feedback: RF only includes the feedback messages, and CF also takes the context (i.e., the question and schema) into account. In FS, the entire conversation is summarized into insights. Finally, EQQ only includes the input question and the final SQL statement.

**Table 1: Execution accuracies of different insight extraction methods, without combining insight extraction methods. *Extracted Question-Query Pairs* are quite powerful since the final SQL statement is a result of all feedback messages of a conversation. The improvements of the other insight extraction approaches, such as *Feedback Summaries*, are negligible but can be combined with *Extracted Quesion-Query Pairs* for further improvements as shown in Table 2.**

| Data set | Model | No Feedback | Hardcoded Few-Shot | naive Conv. History | Extracted QQ-Pairs (EQQ) | Raw Feedback (RF) | Contextualized Feedback (CF) | Feedback Summaries (FS) |
|---|---|---|---|---|---|---|---|---|
| Bird | Claude 3 Sonnet | 44 | 43.8 | 47.8 | **48.5** | 43.8 | 42.8 | 44.5 |
| | Claude 3 Haiku | 38.6 | 43.1 | 42.3 | **46** | 41.2 | 42.6 | 39.5 |
| Fiben | Claude 3 Sonnet | 34.2 | 34.2 | 35.6 | **39.7** | 32.9 | 34.2 | 30.1 |
| | Claude 3 Haiku | 23.3 | 30.1 | 26.0 | **38.4** | 23.3 | 23.3 | 13.7 |
| Fleet-Sweep | Claude 3 Sonnet | 12.8 | 14.7 | 17.3 | **19.9** | 16.7 | 16.7 | 14.1 |
| | Claude 3 Haiku | 14.1 | 13.5 | 14.7 | **18.6** | 10.9 | 16 | 12.2 |

## 2.2 Retrieving Relevant Insights

After we extracted the relevant insights from the conversation history, we need to make use of them when a new question enters the system. Thus, we need to efficiently select those insights that are most relevant to the new question, such that we can put them in the prompt for the Text-to-SQL translation. Hence, this task is similar to schema-linking [15], which is the task of selecting relevant tables and columns to put in the Text-to-SQL prompt. Therefore, similar to recent work [19], we use a Large Language Model to decide which extracted insights are most relevant. However, since the pool of extracted insights can be arbitrarily large, leading to long prompts and high computational overhead, we combine the LLM selection with a dense retriever as explained below and depicted in Figure 3.

**Dense insight retrieval.** In big cloud systems, such as Amazon Q Generative SQL [1], we expect the history of past conversations to become very large. Thus, similar to schema-linking, where there are many columns in the schema to retrieve from, we are faced with a similar problem where there could be very many insights to retrieve from (multiple per past conversation). Therefore, the naive solution of prompting a Large Language Model with all extracted insights to let it select the most important ones is not cost-effective as the dollar cost of using today's LLMs typically increases linearly with the number of input tokens. Similar to related work that aims for cost-effective schema linking [8], we employ a two-staged approach where we combine a dense retriever based on Approximate Nearest Neighbor (ANN) search and semantic similarity with LLM prompting. More specifically, we first embed all past input questions using Amazon Titan Text Embeddings [2] and then index them using FAISS [5]. Once a new question comes in, we can compute the embedding of the new question and use FAISS to retrieve the semantically most similar questions from the conversation history. Additionally, we use a hash map to link all previous questions to the insights extracted from the conversations initiated by them. Thus, we can efficiently gather all insights from questions similar to the current one. Note that since we are retrieving based on the embedding of past questions and not insights, our retrieval procedure eliminates the need to tune insight extraction for retrieval.

**LLM insight selection.** Dense insight retrieval, since it is only based on embedding similarity, will result in many false positives. To make sure that only truly relevant insights are selected for a new question, we use a Large Language Model to make the final decision, similar to related work [8, 19]. As depicted in Figure 3 (right), we put all previously retrieved feedback items into a prompt and instructed a Large Language Model to output a list of the most relevant insights. This results in a variable number of selected insights per question depending on how many insights the LLM considers relevant. Similar to related work from schema-linking [19], we use chain-of-thought [22] and let the LLM reason about the insights before making the decision. We use all selected insights for the Text-to-SQL translation in the next step.

# 3 Evaluation

In our evaluation, we aim to answer the question of whether our two-staged extract-then-retrieve approach can effectively make use of the conversation history in order to improve the accuracy of translation for new questions. We find that using the conversation history can indeed boost the execution accuracy by up to 14.9%, when we combine different insight extraction approaches.

## 3.1 Evaluation Setup

**Data sets.** We use three different data sets. For each data set, we use 30% of questions for evaluation and 70% as our conversation history with user feedback obtained as described in Section 3.2.

(1) **Bird** [12] is a popular Text-to-SQL benchmark data set. As there is no conversation history in the data set with past questions, we instead generate them as described in Section 3.2. Thus, we cannot use the Bird test set for evaluation since it is kept private, and we, therefore, cannot generate any conversations on the databases from the test set. Instead, we used the 11 development set databases.
(2) **Fiben** [18] is a Text-to-SQL benchmark that comes with a single large database that contains 152 tables. It emulates a real-world data mart and models information about public companies, their officers, and financial metrics. We choose the subset of 245 questions (from the original 300) where the provided ground truth SQL statement runs on SQLite and execution does not time out (5 seconds timeout).
(3) **Fleet-Sweep** is an Amazon-internal Text-to-SQL benchmark that consists of a single database with 150 tables and 500 questions. It originates from an internal database that tracks

Redshift metrics and usage statistics. The SQL queries are collected from the query log, and the corresponding questions are generated using a Large Language Model.

**Models.** In our experiments, we used a smaller and a larger model to learn about how different model capacities impact the effect of user feedback from the conversation history. Specifically, we use the Anthropic Claude 3 Haiku Model [20] as the smaller model and the Claude 3 Sonnet Model [20] as our larger model.

**Metrics.** As a metric, we use Execution Accuracy as it is most common in the Text-to-SQL literature [12, 23]. With Execution Accuracy, the predicted SQL statements can differ from the ground truth provided by the data set. Instead, it measures the fraction of predicted queries that, when executed on the database, return the same result as the ground truth SQL statement.

**Text-to-SQL Pipeline.** Our Text-to-SQL approach is inspired by state-of-the-art methods [19, 8]. For schema linking, we first use Approximate Nearest Neighbor (ANN) search to retrieve potentially relevant tables and columns. Then, we make two LLM calls (using either Claude 3 Haiku or Claude 3 Sonnet): the first selects the relevant tables, and the second determines which columns should be included in the prompt for SQL generation. Then, we put the input question together with the relevant columns and some example values for each column into a prompt and instruct the LLM to generate a SQL query. We use chain-of-thought [22], but no advanced reasoning methods such as self-consistency [21], that are often used to boost accuracy but are orthogonal to the problem of learning from the conversation history.

## 3.2 Obtaining Conversations

Unfortunately, none of the previously presented data sets contain conversations of users with Text-to-SQL systems. Moreover, obtaining them manually by letting humans converse with Text-to-SQL systems comes with a prohibitively large overhead. Thus, we generate the necessary conversations by letting two Large Language Models, one for generating the SQL statements and one for generating feedback, have a conversation with each other. More specifically, we take a question from a data set, generate a prompt for Text-to-SQL, and let the LLM for SQL generation (i.e., either Claude 3 Sonnet or Haiku) generate the first SQL statement. We then feed this SQL statement together with the input question and the linked schema into the LLM for feedback generation to generate some feedback. However, we found that for feedback generation, using the same LLM with the same information as for SQL generation does not lead to helpful feedback. Thus, we choose a model from a different model family than the Claude family of models, which has been trained differently and thus has different strengths and weaknesses. We also make sure that we choose a model with high capacity (i.e., many parameters) to simulate the user's advanced data understanding capability. For these reasons, we choose Llama 3.1 405B [6] for feedback generation. Additionally, we provide the LLM for feedback generation with the ground truth SQL and the ground truth query result. Thus, it is able to give relevant feedback by simply comparing ground truth with predicted SQL query.

**Setting.** As explained in the introduction, we assume the existence of an `accept` button that users click once they are satisfied with the generated SQL statement. For the presented feedback generation procedure, we assume the user clicks this button once a SQL statement is generated that is equivalent to the ground truth provided by the data set (according to Execution Accuracy). Since some questions are still difficult to translate even with provided feedback, we limit the number of iterations of the procedure to five (i.e., at most five SQL statements are generated per question) and assume the user left the conversation without clicking the `accept` button if the correct SQL statement was not generated in these five iterations. In Experiments 1 and 2, we only use the extracted insights from those conversations where the user accepted the final SQL statement. In Experiment 3, we investigate the robustness of the approaches to insights extracted from unsuccessful conversations (e.g., when a user accidentally marks an incorrect SQL statement as correct).

## 3.3 Exp. 1: Insight Extraction

Table 1 shows the results of the different insight extraction methods on the three data sets. We use our LLM-Insight-Retriever for *Extracted Question-Query Pairs* (EQQ), *Raw Feedback Messages* (RF), *Contextualized Feedback Messages* (CF) and *Feedback Summaries* (FS). *No Feedback* is the baseline where only the question and the relevant columns (provided by schema-linking) are given as input. Additionally, the table shows two more baselines: *Hardcoded Few-Shot* is similar to *Extracted Question-Query Pairs* in that we add some few-shot examples to the prompt. The difference is that for *Hardcoded Few-Shot*, we assume the administrator selects five few-shot examples (randomly, from the ground truth), while *Extracted Question-Query Pairs* has access to all final SQL statements in the conversation history and retrieves them using our insight retrieval pipeline. Thus, we can show the difference between hardcoding few-shot examples and utilizing the conversation history to obtain few-shot examples. The other baseline is the *naive* usage of the *conversation history*, where we simply prepend all the conversation history to the Text-to-SQL prompt. However, since prompts can become too large this way, we limit the number of conversations we prepend such that the prompt contains at most 100,000 tokens.

We see that using the conversation history naively already results in an improvement of +3.8% on Bird and +4.5% on Fleet-Sweep when using Claude 3 Sonnet. However, using EQQ consistently improves over using conversations, even though conversations contain both the question and the final SQL statement. We suspect that this is due to the lost-in-the-middle effect [13] where LLMs are only good at using the information that is at the beginning and end of a very long prompt and have problems using the information in the middle of the prompt. Moreover, some relevant information might also be cut off due to the token limit. EQQ, on the other hand, leads to very short prompts (only a few few-shot examples are added instead of up to 100,000 tokens), making it easy for the LLM to use the information and avoids any context cut-off. For EQQ, we can see consistent improvements across all data sets and models. For instance, on Bird using Claude 3 Sonnet, we see an improvement of +4.5%, which is 0.7% more than when just using conversations. In particular, EQQ consistently outperforms *Hardcoded Few-shot Examples*, highlighting the benefit of having access to a wide variety of Question-Query pairs as given by the conversation history and carefully selecting them using our insight retrieval procedure.

**Table 2: Combining *Extracted Question-Query Pairs* (EQQ) with *Feedback Summaries* (FS) provides consistently the most improvement when used with the larger Claude 3 Sonnet model. Here, we show the improvements over No-Feedback.**

| Data set | Model | EQQ | EQQ + CF | EQQ + FS |
|---|---|---|---|---|
| Bird | Sonnet | + 4.5 | + 4.7 | **+ 6.2** |
| | Haiku | + 7.4 | + 7.2 | **+ 7.6** |
| Fiben | Sonnet | + 5.5 | + 5.5 | **+ 6.8** |
| | Haiku | + 15.0 | **+ 20.5** | + 12.3 |
| Fleet-Sweep | Sonnet | +9.2 | + 10.3 | **+14.9** |
| | Haiku | + 9.6 | **+ 11.9** | + 5.7 |

Perhaps surprisingly, all other approaches, such as RF, CF, and FS, have a mixed effect on execution accuracy. While we observe small improvements for some data sets, we also see accuracy drops for others. For instance, when using CF on Fleet-Sweep with Claude 3 Sonnet, we see an improvement of +3.9% over no-feedback. However, we also see large drops, e.g., when using FS on Fiben. Moreover, EQQ consistently outperforms all other feedback approaches. Thus, it seems that it is harder for the LLM to extract relevant information from the textual insights (RF, CF, FS) than from few-shot examples, and sometimes, it is even misled by the textual insights. However, as we will see next, textual insights and EQQ seem complementary and can thus be combined to achieve even higher improvements.

### 3.4 Exp. 2: Combining Extraction Approaches

For this experiment, we use the previous experiment's finding that EQQ is extremely strong. As explained before, this is likely due to the fact that the final SQL query essentially captures the information of the entire conversation since it incorporates all user feedback. Hence, we re-run the experiment, but now we combine RF, CF and FS with EQQ. Interestingly, as shown in Table 2, we can now observe consistent improvements, even if some of the approaches led to accuracy drops without few-shot samples. We observe the largest increase of +14.9% on Fleet-Sweep when using FS and Claude 3 Sonnet. This is 5.7% higher than just using EQQ. Another interesting finding is that the best feedback method seems to depend on the size of the model. While for Claude 3 Sonnet, FS with EQQ is the consistently best approach, this picture is not so clear for Claude 3 Haiku. In fact, on Fiben, we even see a small drop compared to just EQQ. Instead, CF performs best with Haiku. Here, we see improvements across the board compared to EQQ, for instance, +2.3% on Fleet-Sweep. RF combined with EQQ (not shown) performs consistently worse than or equal to the shown feedback combinations. To summarize, textual insights, such as *Feedback Summaries* and *Contextualized Feedback* seem to only bring consistent improvements when combined with few-shot examples from *Extracted Question and Query Pairs*. However, the improvements of that combination can be quite large, e.g., up to 14.9% on Fleet-Sweep compared to no feedback. Thus, it seems that for FS and CF to be effective, there need to be relevant few-shot examples from EQQ in the prompt, which is essential for understanding the data and avoiding simple mistakes.

### 3.5 Exp. 3: Robustness to User Mistakes

In the previous experiments, we assumed that users click the `accept` button if and only if they are satisfied with the final generated SQL statement (i.e. if it is equivalent to the ground truth). This ensures, for instance, that for EQQ, only correct SQL statements will be added as few-shot examples to prompts for new questions. In this experiment, we examine if using extracted insights from the conversation history is robust to relaxing this assumption (i.e., users click `accept` even if the generated SQL statement does not equal the ground truth). More precisely, in this experiment, we assume that users click on `accept` blindly after providing feedback only once, leading to SQL statements being accepted by mistake. Perhaps surprisingly, most of the conclusions that can be drawn from previous experiments still hold in this scenario as well. In particular, using EQQ is still effective, leading to an improvement of +1.4% when using Claude 3 Sonnet on Bird (+7% for Claude 3 Haiku). We hypothesize that this is the case because, even when some very question-specific details are wrong in some retrieved few-shot examples, they still incorporate one round of user feedback and are thus still helpful in avoiding some common mistakes. However, the overall improvements are significantly lower than before. For instance, previously, we observed an improvement of +4.5% on Bird with Claude3 Sonnet. Again, as before, combining *Extracted Question-Query Pairs* with other types of insights achieves the best results. For instance, on Bird using Claude3 Sonnet, *Feedback Summaries* with extracted Few-Shot examples leads to a total improvement of +3.3% (for Claude 3 Haiku, CF works best with an improvement of +7.5% over no-feedback).

## 4 Limitations and Future Work

In these experiments, we showed that using user feedback from the conversation history of users with Text-to-SQL systems can greatly improve the translation quality for new questions. However, for our analyses, we used LLM-generated feedback. Hence, one interesting question would certainly be how real human feedback differs from LLM-generated feedback. In particular, it remains to be seen if the presented approaches perform as well on human feedback as they do on LLM feedback. Hence, an important next step is to test these approaches with conversations of real users. For further research in this direction, a new crowd-sourced benchmark is needed for conversational Text-to-SQL systems that can be divided into conversation history and new questions.

## 5 Conclusion

In this paper, we showed that user feedback from a conversation history can be used to make Text-to-SQL substantially more accurate. In fact, we have seen improvements of up to 14.9% on some data sets. Moreover, we compared several different approaches to extract different insights from a conversation history. We showed that combining Feedback Summaries or Contextualized Feedback with extracted few-shot examples works best in the majority of cases. However, there are still some open questions. Most importantly, it is unclear whether real human feedback is of similar quality to artificially generated feedback used for the analyses in this paper.

## References


[1] [n. d.] Amazon Q Generative SQL in Redshift: revolutionizing data analysis with ai-powered query generation. https://aws.amazon.com/awstv/watch/85049b8ca5e/. Accessed: 2025-02-21. ().

[2] [n. d.] AWS AI Service Cards: amazon titan text embeddings. https://docs.aws.amazon.com/pdfs/ai/responsible-ai/titan-text-embeddings/titan-text-embeddings.pdf. Accessed: 2025-02-21. ().

[3] Tom Brown et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33, 1877–1901.

[4] Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. Rsl-sql: robust schema linking in text-to-sql generation. *arXiv preprint arXiv:2411.00073*.

[5] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. arXiv: 2401.08281 [cs.LG].

[6] Abhimanyu Dubey et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

[7] Avrilia Floratou et al. 2024. Nl2sql is a solved problem... not! In *CIDR*.

[8] Yingqi Gao et al. 2024. Xiyan-sql: a multi-generator ensemble framework for text-to-sql. *arXiv preprint arXiv:2411.08599*.

[9] Fangyu Lei et al. 2024. Spider 2.0: evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*.

[10] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The dawn of natural language to sql: are we fully ready? *arXiv preprint arXiv:2406.01265*.

[11] Haoyang Li et al. 2024. Codes: towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2, 3, 1–28.

[12] Jinyang Li et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

[13] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: how language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12, 157–173.

[14] Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The death of schema linking? text-to-sql in the age of well-reasoned language models. *arXiv preprint arXiv:2408.07702*.

[15] Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36, 36339–36348.

[16] Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*.

[17] Mohammadreza Pourreza et al. 2024. Chase-sql: multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*.

[18] Jaydeep Sen et al. 2020. ATHENA++: natural language querying for complex nested sql queries. *Proc. VLDB Endow.*, 13, 11, 2747–2759.

[19] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*.

[20] [n. d.] The claude 3 model family: opus, sonnet, haiku. In https://api.semanticscholar.org/CorpusID:268232499.

[21] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

[22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35, 24824–24837.

[23] Tao Yu et al. 2018. Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.


Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009