Speech-to-SQL Parsing: Error Correction with Multi-modal Representations

Anonymous ACL submission

Abstract

We study the task of spoken natural language to SQL parsing (speech-to-SQL), where the goal is to map a spoken utterance to the corresponding SQL. Existing work on SQL parsing has focused on text as input (text-to-SQL). To develop a speech-to-SQL parser, we harness progress in text-to-SQL parsing, and automatic speech recognition (ASR). However, ASR is still error-prone, we therefore propose an error correction method that fixes ASR errors in the context of a DB schema. We present a novel multi-modal representation of text, audio, and DB schema with audio attention and a phoneme prediction auxiliary task. Our experiments show that our method yields better performance, is much faster to train, has greater transparency, and is parser-agnostic compared to baselines that seek to adapt to ASR errors.

1 Introduction

Interfaces that support human language as a medium of communication between humans and computers have been of interest for decades (Winograd, 1971; Woods, 1972; Codd, 1974; Hendrix et al., 1978; Zelle and Mooney, 1996; Popescu et al., 2003; Zettlemoyer and Collins, 2012). Known as Natural Language Interfaces (NLIs), early systems saw limited success due to the difficult problem of endowing computers with the ability to understand natural language. Progress in language understanding has led to renewed interest in NLIs. In particular, several studies have focused on NLIs to databases (NLIDBs) (Zhong et al., 2017; Yin and Neubig, 2017; Yu et al., 2018; He et al., 2019; Guo et al., 2019; Wang et al., 2020). NLIDBs, when fully realized, stand to support users who are not proficient in query languages.

Motivation. The primary focus of NLIDBs has been on parsing natural language *text* utterances into executable SQL queries (text-to-SQL parsing). Motivated by the rise of speech-driven digital assistants on smartphones, tablets, and other small handheld devices, we study the task of parsing spoken natural language to executable SQL queries (speech-to-SQL parsing). A speech-to-SQL parser has a number of potential use cases. For example, in the healthcare domain, a nurse practitioner at a patient bedside typically looks up patient details on a desktop in the patient's room by filling out forms whose back-end is a database, where speech-to-SQL could be used instead, for faster results. Furthermore, speech-to-SQL removes the need for keyboards that can be slow and cumbersome on small devices, when querying databases.

Approach. To build a speech-to-SQL parser, we leverage progress in text-to-SQL parsing, automatic speech recognition (ASR). However, ASR is still error-prone. To deal with ASR errors, we propose an error correction method that fixes ASR errors in the context of a DB schema. Our error correction method, *TaggerILM rewriter*, edits the ASR transcription by tagging tokens (Tagger) to indicate if they should be edited, and then rewriting the appropriate tokens using an infilling language model (ILM). We build both the Tagger and ILM on top of a novel multi-modal representation of text, audio, and DB schema, with audio attention and a phoneme prediction auxiliary task.

Summary of Contributions. We make the following contributions: i) study the problem of spoken natural language to SQL parsing, illustrated in Figure 1, which is currently under-explored in the neural era. ii) propose an ASR error correction method, TaggerILM rewriter build on a novel encoder that produces multi-modal representations from text, audio, and DB schema representations using audio attention and a phoneme prediction auxiliary task. We show that the TaggerILM rewriter yields better performance, is faster to train, has greater transparency and is parser-agnostic. iii)

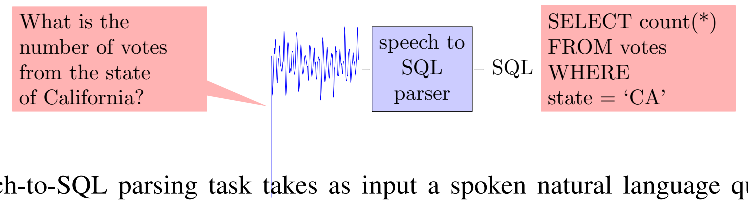

What is the number of votes from the state of California?
speech to SQL parser
SQL
SELECT count(*) FROM votes WHERE state = 'CA'

Figure 1: The speech-to-SQL parsing task takes as input a spoken natural language query, and outputs the corresponding SQL query.

Gold:	Show	me	the	fares	from	Dallas	to	Boston
ASR:	Show	me		affairs	from	Dallas	to	Boston

Figure 2: An example ASR error wherein a phrase, "the fares", is incorrectly described as “affairs". Passing ASR errors to the text-to-SQL parser is unlikely to produce the correct SQL.

present a new dataset which is a spoken version of the Spider text-to-SQL benchmark (Yu et al., 2018), named Spoken Spider (link provided in Appendix). iv) carried out extensive experiments on Spoken Spider, showing the strengths and limitations of our proposed methods via an in-depth analysis to provide guidance for future research in this direction.

2 Baselines

Blackbox Baseline. Given a spoken utterance, the task is to emit the corresponding SQL. An obvious solution to this problem is to first pass the spoken utterance through an automatic speech recognizer (ASR), and then issue the top-ranked ASR transcription to a text-to-SQL parser which produces the final SQL. We name it the *blackbox* baseline. A drawback of this baseline is that no attempt is made to deal with ASR errors. Figure 2 shows an example of such errors. Passing ASR errors to the text-to-SQL parser is unlikely to produce the correct SQL.

Domain Adaptation Baselines. We consider another set of baselines that frame the problem of speech-to-SQL as a domain adaptation problem of text-to-SQL. That is, we can treat clean text as the source domain, and ASR transcriptions as the target domain. The source domain data is the original text-to-SQL data, i.e. clean text and gold SQL queries. We generate the target domain data by pairing ASR transcriptions with gold SQL queries. To let the parser learn to adapt to ASR errors (target domain), we train the text-to-SQL parser model on the new data we generated. Domain adaption via data pre-processing can produce strong results (Daumé, 2007). We consider two variations, *Retraining-ASR* and *Retraining-mixed*. *Retraining-ASR* uses the target domain data only; *Retraining-mixed* uses both the source and the target domain data.

3 Method

We propose a neural error correction method that fixes ASR errors in the transcription before passing the transcription to a text-to-SQL parser. Our proposed method, *TaggerILM rewriter*, consists of a Tagger and an Infilling Language Model (ILM) rewriter. The input to our TaggerILM rewriter consists of: i) the top-k ASR transcriptions. ii) the DB schema. iii) the raw audio stream of input speech and schema tokens. The output is a corrected ASR transcription. In contrast to adaptation baselines, fixing errors has the advantage of transparency, as opposed to implicitly adapting to errors. Furthermore, this approach is agnostic to the text-to-SQL parser, thus can be applied to any new strong parser without requiring additional training of the parser, which is a significant computational advantage.

Tagging Tokens. Our Tagger first tags each token in the input transcription as `KEEP`, `DEL` or `EDIT`. These tags are denoted as *rewriter tags*. The `DEL` and `EDIT` tags are based on the BIOUL tagging schema, thus marking certain *spans*[1] in the sentence to be deleted or edited. The `KEEP` tags are the O tags in BIOUL, marking tokens that should be kept. After the tagging step, tokens marked as `KEEP` or `DELETE` are kept or deleted, respectively. Each `EDIT` span is replaced by a `[BLANK]` token.

Rewriting Spans with an Infilling Language Model (ILM). After tagging, the ILM rewriter, which is an Infilling Language Model (ILM) (Donahue et al., 2020), takes the transformed sentence as input and fills in each `[BLANK]`. The ILM is an autoregressive language model. A working example of ILM is shown in Table 1. It takes as input a sentence with blanks, and predicts a sequence with

[1]For example, tokens from `B-DEL` to `L-DEL`, or single `U-DEL`, make a `DEL` span. See Table 1 for a concrete example.

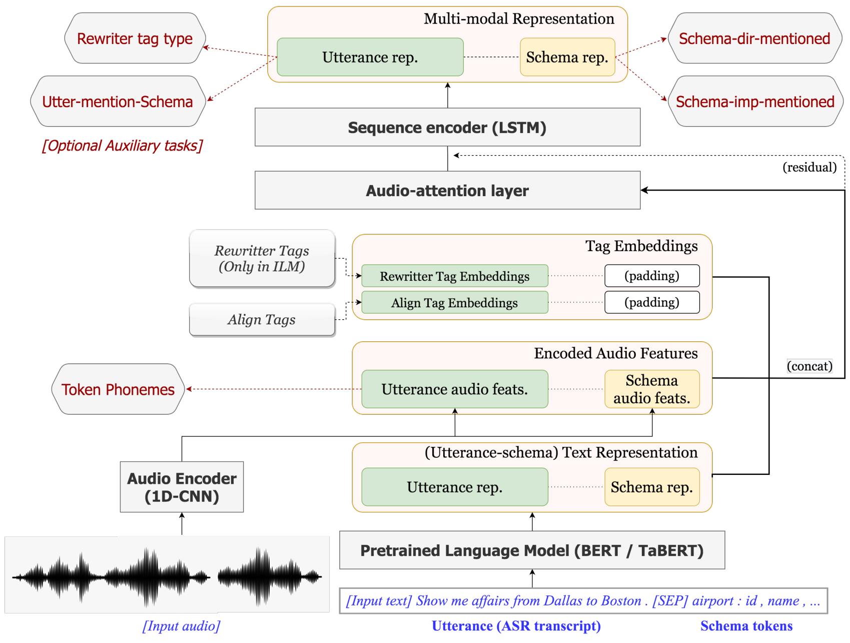


Figure 3: Our error correction encoder fuses free text, audio, and a structured DB schema to obtain a multi-modal representation. Auxiliary tasks such as the "token phonemes" task seek to ensure that certain information is encoded in the representation.

Gold	`Whose  name has       substring     ABC  ?`
ASR	`Who's  name has  a    sub    string ABC  .`
Tags	`U-EDIT KEEP KEEP U-DEL B-EDIT L-EDIT KEEP U-EDIT`
ILM Input	*[BLANK] name has [BLANK] ABC [BLANK]*
ILM Prediction	*Whose [ANS] substring [ANS] ? [ANS]*
Final Output	*Whose name has substring ABC ?*

Table 1: Infilling Language Model (ILM) example. Gold text is invisible to the model. In the tagged ASR output, adjacent tokens with a single `U` or from `B` to `L` make a *span*. For example, “Who’s” and “sub string” are `EDIT` spans and “a” is a `DEL` span.

content for each blank. Content for different blanks are separated by a special `[ANS]` token.

We next describe the details of the Tagger and the ILM rewriter. They share an encoder, but each model has a separate decoder.

3.1 Shared Encoder

We introduce a novel encoder that fuses different representations from free text, audio, and structured DB schema, illustrated in Figure 3. The novel components of the shared encoder include an *audio attention layer*, *align tag* features, and *auxiliary probing loss*. This encoder is shared by both the Tagger and the ILM rewriter, with only slight variations to accommodate differences in their input. We introduce each part of the model in detail.

3.1.1 Input Modalities and Features

Text. We obtain the representation of the utterance and schema text using a pre-trained language model (PLM), as shown in Figure 3, for transfer learning of language. A possible choice is to utilize a general-purpose PLM, such as BERT. We concatenate an utterance (ASR transcription) and DB schema with a [SEP] token and feed the concatenated sequence into the PLM to get a contextualized utterance-schema representation for each token (using a scalar mix of the hidden representations from each self-attention layer). We also experimented with TaBERT (Yin et al., 2020), a BERT-style model that jointly represents a natural language sentence and structured data in a DB. The details of adapting TaBERT to our model are in the appendix.

Audio. Our input includes the audio features of both utterance tokens and schema tokens. For the utterance, we extract the audio slice of each token using the timestamps provided in ASR output. For schema tokens, we directly use a speech synthesizer to obtain their audios. We use a standard 1D-CNN to obtain audio encoding vectors for each token[2].

Tags. There are two types of tags used in our encoder: *align tags* and *rewriter tags*. First, *align tags* provide information about whether other ASR candidates agree with the input one on a certain token. If all other ASR candidates agree, the token is tagged *[SAME]*; if x other candidates agree but y disagree, the tag will be *[DIFF(d)]* where $d = x - y$. Intuitively, these tags include additional information from other ASR candidates to help the model decide the correctness of each token, which is helpful given that the input text of our model includes only one ASR candidate. Second, we have *rewriter tags* which flag tokens to be kept or modified. These tags will be the output of Tagger and input of ILM rewriter. Both types of tags are embedded into vectors using a separate randomly-initialized embedding table.

3.1.2 Multi-modal Representation

To generate a final representation of the input modalities and features, we concatenate the text features, audio features and tag embedding features of each token, and feed the result into an *audio-attention* layer, followed by a standard LSTM sequence encoder, to get the multi-modal representation. The audio-attention layer is similar to a standard self-attention layer, but only uses audio features as attention keys and queries to compute attention weights instead of using the whole concatenated representations. We use cosine attention, and add the token features to the attention output as a residual link. The multi-modal representation is then used by the decoder of different models, which we describe later.

3.1.3 Encoder Probes as Auxiliary Losses

Since the encoder is a complex combination of different modalities, there is potential to lose important predictive information in the training process. Thus, to encourage representations to retain certain important information, we added corresponding probes to provide guidance for the internal representations of the model. The auxiliary losses of probes are added to the total training loss. Below, *Joint* probes are applied onto the the multi-modal joint representation, and *Audio* probes are applied on the audio encoding.

[2]Details are given in appendix.

A) Rewriter tags (Joint). predicting the type of rewriter tag (`KEEP/EDIT/DEL`) for each token. This is similar to the task for Tagger but without BIOUL prefixes.

B) Utterance token mentions schema (Joint). predicting whether an utterance token refers to a schema item, i.e. a table or column. The purpose is to force schema information to be expressed in the utterance representation.

C) Schema item directly mentioned (Joint). predicting whether a schema item is directly mentioned by the utterance. A direct mention essentially means a string match. This is to ensure the utterance information is fused into schema representation.

D) Schema item implicitly mentioned (Joint). predicting whether a schema item is implicitly mentioned by the utterance. An implicit mention means the item appears in the final SQL query. This is more challenging than task C, and encourages a deeper understanding in the multi-modal representation towards the final SQL prediction task.

E) Token phonemes (Audio). predicting which phonemes exist in the token pronunciation. This is a multi-label classification task. This task is applied to the audio encoding only, since language understanding should not be needed for predicting phonemes. As the input ASR transcription may contain errors, this task enforces the model to utilize the raw audio stream and not fully rely on the text modality.

We studied adding each auxiliary task separately. Each task involves an MLP classifier head on top of the audio or joint representations.

3.2 Decoders

For both Tagger and ILM, the decoder takes the multi-modal representation from encoder as input.

Tagger Decoder. The Tagger decoder is an LSTM-CRF sequence labeler. During training, we require "gold rewriter tags" to supervise the Tagger. We leverage work on aligning tokens in machine translation. In particular, we apply Fast Aligner (Dyer et al., 2013) to every ASR candidate and its corresponding ground truth text and obtain the gold tags for each token based on the output alignment.

Infilling LM Decoder. The ILM rewriter decoder is a standard LSTM-based decoder. During training, it takes the gold rewriter tags as input, to avoid error cascading from the Tagger. The gold output is the correct blank-filling sequence, described above. During inference, it takes the Tagger prediction of rewriter tags as input.

4 Experiments

4.1 Experimental Setup

Dataset. The main dataset we use is Spider (Yu et al., 2018). Spider is a large-scale text-to-SQL dataset in which the train, dev, and test data have a different subset of DBs, thus models must generalize to unseen databases. In order to evaluate speech-to-SQL systems, we created a spoken version of Spider, named Spoken Spider. We used Amazon Polly speech synthesizer to obtain the audio of all natural language queries and tokens in DB schemata.

External models. Our baselines and proposed methods make use of an external ASR system and text-to-SQL system. For ASR, we use Amazon Transcribe, which is a state-of-the-art commercial ASR system whose outputs includes the transcription candidate list. It also outputs the timestamps and confidence scores of each transcribed token in each candidate. For text-to-SQL parsing, we mainly use RAT-SQL (Wang et al., 2020) which was one of the best-performing models on the Spider leaderboard when we started our experiments. Currently it is still a key part of many competitive methods on the leaderboard. In order to show the advantage of our TaggerILM rewriter being parser-agnostic, we also experimented with a stronger text-to-SQL parser, Picard, which was proposed very recently at EMNLP 2021 (Scholak et al., 2021). We used the released Picard model checkpoint with no further training. Due to computational resource limitation, we did not retrain the Picard model.

Evaluation Metrics. For evaluation, we use BLEU score against the gold utterance as the metrics for text quality. To measure the end-to-end speech-to-SQL performance, we use the SQL exact match score provided in the Spider official evaluation script, which is the metrics for one of the Spider leaderboards.

4.2 BLEU, Exact Match, and Run Time

The main results are shown in Figure 4. For our TaggerILM method, we trained each model (Tagger & ILM) 5 times, only varying the random seeds between runs. The bar heights represent the average values and the error bar lengths represents the standard deviations.

With RAT-SQL Parser. With RAT-SQL as the text-to-SQL parser, our TaggerILM rewriter significantly outperforms the blackbox baseline on both BLEU (0.871 vs 0.801) and SQL exact match scores (0.528 vs 0.455). Compared to retraining methods, TaggerILM has the same significant advantage in BLEU, as retraining methods make no efforts to revise the text directly. On exact match scores, TaggerILM method has a slight advantage over retraining methods, although the improvement is not statistically significant ($p \simeq 0.15$). Besides performance differences, retraining methods have several limitations. First, retraining the text-to-SQL parser is computationally expensive. As shown in Figure 4b, retraining methods take significantly longer to train, compared to TaggerILM, even when we used a small parser model[3]. Furthermore, in real-world applications, users usually want to check the text input to ensure that the system has understood the spoken utterance. Not being able to fix the text could hurt user trust of the system.

With Picard Parser. Using the recent parser, Picard, the exact match score is boosted for Blackbox baseline to outperform any method with RAT-SQL. Nonetheless, applying our TaggerILM further significantly improves the performance. These results with Picard reflect the parser-agnostic nature of our TaggerILM rewriter where our text-to-SQL parser can be quickly swapped out for another without further training.

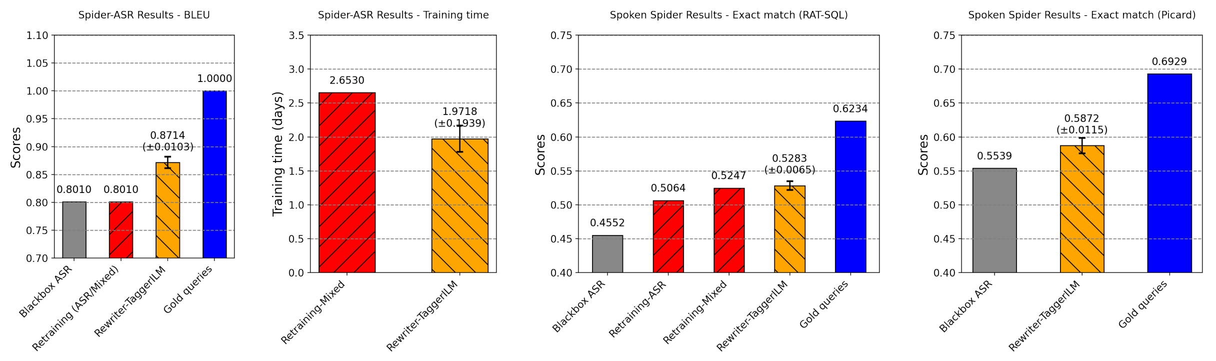
Lastly, there is large performance gap between all methods and directly passing the gold queries to text-to-SQL parser. This shows room for further improvements on the task of speech-to-SQL.

4.3 Analysis

4.3.1 Oracle Replacements

To examine the bottleneck of our method, we tested the performance of the Tagger and ILM rewriter separately, with the other part replaced by an oracle. In detail, the oracle Tagger always predicts the gold labels for rewriter tags; the oracle ILM rewriter always rewrites an edit span with its aligned span in the gold utterance.

[3]We used the smaller model, RAT-SQL-Glove, instead of the best performing RAT-SQL-BERT, due to GPU memory limit.



(a) BLEU Scores. (b) Training time. (c) Exact Match Scores (RAT-SQL). (d) Exact Match Scores (Picard).

Figure 4: Spoken Spider results. **(a)** BLEU scores are computed on the text transcriptions. **(b)** training time for Blackbox ASR and Gold queries are not reported as we used out-of-the-box text-to-SQL model and additional training is not required. **(c)** SQL exact match using RAT-SQL as parser. **(d)** SQL exact match using Picard as parser. Since we were not able to retrain the recently released Picard, we do not present Picard results for baselines that require retraining.

Tagger	ILM	Exact Match	BLEU
Trained	Trained	0.5283(±0.0065)	0.8714(±0.0103)
Oracle	Trained	0.5262(±0.0079)	0.8951(±0.0044)
Trained	Oracle	0.5755(±0.0051)	0.9246(±0.0027)

Table 2: Oracle analysis results showing that the ILM is the current performance bottleneck.

The results are in Table 2. Using an oracle tagger only gives minor improvement on BLEU and no improvement for exact match. However, using an oracle ILM rewriter provides a significant performance boost on both metrics. We can therefore conclude that ILM rewriter is the bottleneck in the TaggerILM pipeline. In future work, to further improve the TaggerILM method (or the like), the key would be to improve the ILM rewriter part.

4.3.2 Syntactic Category Analysis

To better understand the strengths and limitations of our approach, we analyzed the token accuracy of rewritten utterances on each Part-of-Speech (POS) tag. The results are shown in Table 3, under meta-column “Token Accuracy”. POS tags are sorted by ΔAcc, i.e. the token accuracy change of rewritten utterances compared to raw ASR output. At the bottom are the POS tags on which the rewriter is least successful. The PROPN (proper noun) is the hardest POS tag for the rewriter. Most proper nouns in the Spider dataset are value literals in databases. They are challenging for the ASR system because they are mostly uncommon words that are unlikely to appear in ASR training data. Additionally, our current pipeline does not provide synthesized audio for value literals because of the excessively large amount of them. It would be an important future work to improve the rewriting performance on these value literals, potentially by better representing and combining the audio features of utterance tokens and the database tokens.

POS tags	Total	Token Accuracy			Exact Match	
		Raw	Rewritten	ΔAcc↑	ΔF↓	ΔM↑
PUNCT	605	0.6661	0.8529	0.1868	-0.0183	0.0201
NUM	114	0.6228	0.7719	0.1491	-0.0128	0.0146
PRON	316	0.9494	0.9935	0.0441	-0.0110	0.0146
VERB	508	0.9134	0.9567	0.0433	-0.0018	0.0055
AUX	519	0.9364	0.9595	0.0231	0.0018	-0.0018
NOUN	1875	0.9275	0.9482	0.0207	-0.0164	0.0146
ADP	824	0.9551	0.9757	0.0206	-0.0091	0.0110
DET	1093	0.9607	0.9725	0.0118	-0.0018	0.0073
SCONJ	55	0.9636	0.9636	0.0000	0.0000	0.0018
ADJ	489	0.9796	0.9775	-0.0021	0.0000	0.0018
CCONJ	176	0.9886	0.9830	-0.0056	0.0000	0.0018
ADV	175	0.9486	0.9429	-0.0057	0.0037	-0.0018
PART	63	0.8730	0.8571	-0.0159	0.0000	0.0018
PROPN	276	0.7138	0.6436	-0.0702	-0.0091	0.0183

Table 3: TaggerILM performance analysis with freezing/modifying POS tags. “Raw” stands for raw ASR output; “Rewritten” for utterances rewritten by our model. The proper nouns (PROPN) tag is the most challenging for our model. ΔF is the exact match performance drop when freezing tokens under the POS (the lower, the edits more helpful). ΔM is the performance gain when only modifying the POS (the higher the more helpful).

To further examine the separate influence of each POS on the SQL exact match performance, we experimented to freeze tokens with certain POS during ILM rewriting, or only modifying tokens with certain POS, and check the performance changes. Conceptually, a higher performance loss when freezing or a higher gain when modifying a POS indicates that rewriting this POS provides more positive influence on the final performance. The results are in Table 3, under “Exact Match”. The most contributing POS include PUNCT, NUM, PRON, NOUN, ADP (adpositions, such as “in”, “and”, “or”, etc.) and PROPN. NOUN is important as expected, because most of the entity mentions that are directly related to SQL query are nouns. It is also the most frequency POS in Spider. Surprisingly, fixing PROPN has a highly positive affect on the SQL performance, although its token accuracy largely dropped. A possible explanation is that, since the exact match metrics ignore value literals in SQL, having an incorrect proper noun with correct type (e.g. “Johnson” and “Jason”) suffices for a correct SQL. Besides, ADP are also important because they often decide the logic operators in the SQL. Other influential POS are less explainable. Our assumption is that fixing these POS conceptually maps the utterance back to the domain where text-to-SQL parser is trained, therefore improves performance. Several examples of correcting certain POS improving SQL prediction are shown in Table 6.

4.3.3 Probing Tests

Aside from quantitative performances, we also check the internal behaviors of the model to shed light on what information is captured in the hidden representation. We do probing tests on the same tasks as mentioned in Section 3.1.3. We examine to what extent adding such auxiliary losses can help the model capture corresponding information, and how much they improve the final performance. We only experiment on ILM rewriter for these experiments, not Tagger, as we have shown that ILM is the overall performance bottleneck. Results are in Table 4. The task of classifying gold rewriter tags (A) achieved F1 score close to 1. This indicates that tag information is already captured, and adding auxiliary loss on this task is unnecessary. For other tasks on utterance-schema relation (B-D), adding corresponding auxiliary loss can drastically improve the probing accuracy. However, they bring no clear improvement on the model performance, either BLEU or exact match. This implies that the model is not bottle-necked by the fusion of utterance and schema features. In contrast, adding a loss for token phonemes (E) improves both probing accuracy and overall performance (on exact match). The probe for token phonemes is similar to an acoustic model; however, before adding the loss, the probing F1 score is very low. As we observe the model, the audio encoder often degrades and output the same encoding for every token, indicating that the model is relying solely on text features and ignoring audio input. As a result, adding the loss on phonemes “activated” the audio encoder and also slightly improved the overall performance.

Task	P(pos)	No Aux Loss			With Aux Loss		
		Probe F1	EM	BLEU	Probe F1	EM	BLEU
(A) Gold tags	0.3480	0.9969	-	-	-	-	-
(B) Utter-MS	0.2573	0.7899	0.5283	0.8714	0.9223	0.5225	0.8730
(C) Schema-DM	0.1113	0.2369	0.5283	0.8714	0.7462	0.5225	0.8710
(D) Schema-IM	0.0713	0.0916	0.5283	0.8714	0.4628	0.5229	0.8714
(E) phonemes	0.0985	0.0221	0.5250	0.8721	0.2113	0.5283	0.8714

Table 4: Probing test results. “P(pos)” is the proportion of positive labels[4]. EM means exact match. Gold rewriter tags (A) are almost fully predicable without explicit supervision. All other tasks (B-E) are more predictable with the auxiliary loss, but only (E) yields a slight improvement in performance.

Ablation	Exact Match	BLEU
Best config	0.5283(±0.0065)	0.8714(±0.0103)
Audio-att→Full-att	0.5159(±0.0107)	0.8705(±0.0090)
- Align tags	0.5214(±0.0144)	0.8705(±0.0107)
BERT→TaBERT	0.4958(±0.0122)	0.8585(±0.0120)

Table 5: Ablation study results showing that our audio attention and align tags in our encoder are important for performance.

4.3.4 Other Ablation Studies

We conducted ablation studies to justify other aforementioned design choices. The results are shown in Table 5. TaBERT is not as good as BERT in our case, even though it is pretrained for table data. It might be because the model becomes less robust to ASR errors after tuning. Other design choices, mainly audio attention and align tags, are also shown to be useful, especially on exact match.

4.4 Sample Predictions

Several sample queries in which the TaggerILM method improved the final SQL prediction are shown in Table 6. These samples illustrate that the rewriter improves SQL accuracy by fixing critical ASR errors, such as “ids” recognized as “It’s” in Table 6(a), “and” recognized as “in” in (b) and “codes” recognized as “coats” in (c). Nonetheless, the rewriter is still unable to fix some errors, for instance, “bought” recognized as “spot” in Table 6(b). To be able to fix such errors, the rewriter will need a better understanding of context and a better audio representation.

ASR	show. It’s for all templates not used by any document.
ASR SQL	SELECT Templates.Template_Details FROM Templates WHERE Templates.Template_ID NOT IN (SELECT Documents.Template_ID FROM Documents) (0.1111)
Rewritten	show ids for all templates not used by any document .
Rewritten SQL	SELECT Templates.Template_ID FROM Templates EXCEPT SELECT Documents.Template_ID FROM Documents (1.0)
Gold	Show ids for all templates not used by any document.
Gold SQL	SELECT template_id FROM Templates EXCEPT SELECT template_id FROM Documents

(a) Fixing pronouns.

ASR	what are the average in maximum number of tickets spot in all visits .
ASR SQL	SELECT Avg(visit.Num_of_Ticket) FROM visit (0.2)
Rewritten	what are the average and maximum number of tickets spot in all visits ?
Rewritten SQL	SELECT Avg(visit.Num_of_Ticket), Max(visit.Num_of_Ticket) FROM visit (1.0)
Gold	What are the average and maximum number of tickets bought in all visits?
Gold SQL	SELECT avg(num_of_ticket) , max(num_of_ticket) FROM visit

(b) Fixing adpositions & punctuation (failed to fix verb).

ASR	what are the coats of template types that are not used for any document?
ASR SQL	SELECT Templates.Template_Details FROM Templates WHERE Templates.Template_ID NOT IN (SELECT Documents.Template_ID FROM Documents) (0.1111)
Rewritten	what are the codes of template types that are not used for any document ?
Rewritten SQL	SELECT Templates.Template_Type_Code FROM Templates EXCEPT SELECT Documents.Template_ID FROM Documents (0.8)
Gold	What are the codes of template types that are not used for any document?
Gold SQL	SELECT template_type_code FROM Templates EXCEPT SELECT template_type_code FROM Templates AS T1 JOIN Documents AS T2 ON T1.template_id = T2.template_id

(c) Fixing nouns.

Table 6: Samples improved by our TaggerILM rewriter. Numbers in brackets after SQL queries are their partial match scores (1.0 is an exact match. Detailed explanations are given in the appendix.)

5 Related Work

Speech-to-SQL and Text-to-SQL. While there is previous work on speech-to-SQL (Jamoussi et al., 2005; Kumar et al., 2013; Hiregoudar et al., 2019), to our knowledge, our work is the first to systematically explored adaption vs. error correction approaches and to propose a method that builds on state-of-the-art deep learning based techniques. A closely related task, SQL dictation (speech-to-text), has recently been introduced (Shah et al., 2020). Our speech-to-SQL system enables users to speak natural language, thus can support users who are not proficient in query languages.

Renewed interest in text-to-SQL parsing has resulted in a number of contributions. New datasets have been introduced, such as WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018). Methods leveraging these datasets have been developed such as NL2Code(Yin and Neubig, 2017), IR-Net (Guo et al., 2019), X-SQL (He et al., 2019), RAT-SQL (Wang et al., 2020), and Picard (Scholak et al., 2021). The methods devised in this paper can benefit from further progress in text-to-SQL parsing to improve speech-to-SQL performance.

ASR Correction. Mani et al. (2020) treat ASR correction as a machine translation (MT) task, but their input only consists of the ASR transcription, no raw audio or DB schema. Weng et al. (2020) leverage multi-task learning by jointly learning a language model (LM) and a dialog state tracker (DST), and using the LM to rerank ASR candidates. Corona et al. (2017) use the confidence of downstream model to rerank ASR transcription candidates. Our rewriter model is more flexible than reranking approaches as it edits the ASR transcriptions.

Spoken Dialogue Systems. Spoken dialogue systems extract semantic concepts from spoken utterances to perform tasks such as intent detection and slot filling. Research on such systems has attacked the problem of adapting to ASR errors, leveraging information from lattices or word confusion networks (Hakkani-Tür et al., 2006; Tür et al., 2013; Ladhak et al., 2016; Zhu et al., 2018; Shivakumar and Georgiou, 2019; Huang and Chen, 2019), simulated errors (Simonnet et al., 2018; Zhu et al., 2018), or by ASR robust contextualized embeddings (Huang and Chen, 2020). Our task on speech-to-SQL has a structured DB schema against which errors can be corrected; however, it is also more challenging due to the complexity of DB schema structures and value literals.

6 Conclusion

We proposed an error correction method for speech-to-SQL parsing. Powered by a novel multi-modal encoder, our proposed TaggerILM method fixes a substantial number of ASR errors as reflected by the strong BLEU scores. Additionally, it significantly outperforms the blackbox baseline on SQL prediction, and outperforms the strong retraining baseline where the gains are significant when we take advantage of our method’s parser-agnostic nature by plugging in a strong newly released text-to-SQL parser, Picard. The in-depth analyses we conducted can serve as a guide for future work towards further improvements on the speech-to-SQL task. For example, future work can explore incorporating more context in the form of literals in the database, to address the current poor performance on proper nouns. Another future direction is to consider an end-to-end fully-differentiable approach to the problem since performance of our current approach is upper-bounded by the performance of the text-to-SQL parser.

References

Edgar F Codd. 1974. Seven steps to rendezvous with the casual user. IBM Corporation.

Rodolfo Corona, Jesse Thomason, and R. Mooney. 2017. Improving black-box speech recognition using semantic parsing. In IJCNLP 2017.

Hal Daumé. 2007. Frustratingly easy domain adaptation. In ACL.

Chris Donahue, Mina Lee, and Percy Liang. 2020. Enabling language models to fill in the blanks. In ACL, pages 2492–2501. Association for Computational Linguistics.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of ibm model 2. In HLT-NAACL.

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. In ACL, pages 4524–4535.

Dilek Hakkani-Tür, Frédéric Béchet, Giuseppe Riccardi, and Gokhan Tur. 2006. Beyond asr 1-best: Using word confusion networks in spoken language understanding. Computer Speech & Language, 20(4):495–514.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019. X-sql: reinforce schema representation with context. arXiv preprint arXiv:1908.08113.

Gary G Hendrix, Earl D Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. 1978. Developing a natural language interface to complex data. ACM Transactions on Database Systems (TODS), 3(2):105–147.

Shravankumar Hiregoudar, Manjunath Gonal, and Karibasappa K G. 2019. Speech to sql generator-a voice based approach. Journal of Basic and Applied Research International, Vol 4:01–05.

Chao-Wei Huang and Yun-Nung Chen. 2019. Adapting pretrained transformer to lattices for spoken language understanding. In IEEE Automatic Speech Recognition and Understanding Workshop, ASRU, pages 845–852.

Chao-Wei Huang and Yun-Nung Chen. 2020. Learning asr-robust contextualized embeddings for spoken language understanding. In ICASSP, pages 8009–8013.

S. Jamoussi, Kamel Smaïli, and J. Haton. 2005. From speech to sql queries : a speech understanding system. In AAAI 2005.

S. Kumar, A. Kumar, P. Mitra, and G. Sundaram. 2013. System and methods for converting speech to sql. ArXiv, abs/1308.3106.

Faisal Ladhak, Ankur Gandhe, Markus Dreyer, Lambert Mathias, Ariya Rastrow, and Björn Hoffmeister. 2016. Latticernn: Recurrent neural networks over lattices. In Interspeech, pages 695–699.

Anirudh Mani, Shruti Palaskar, Nimshi Venkat Meripo, Sandeep Konam, and F. Metze. 2020. Asr error correction and domain adaptation using machine translation. ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6344–6348.

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, pages 149–157.

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.

Vraj Shah, Side Li, Arun Kumar, and Lawrence Saul. 2020. Speakql: Towards speech-driven multimodal querying of structured data. In SIGMOD, pages 2363–2374.

Prashanth Gurunath Shivakumar and Panayiotis G. Georgiou. 2019. Confusion2vec: towards enriching vector space word representations with representational ambiguities. PeerJ Comput. Sci., 5.

Edwin Simonnet, Sahar Ghannay, Nathalie Camelin, and Yannick Estève. 2018. Simulating ASR errors for training SLU systems. In LREC.

Gökhan Tür, Anoop Deoras, and Dilek Hakkani-Tür. 2013. Semantic parsing using word confusion networks with conditional random fields. In INTERSPEECH, pages 2579–2583.

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 ACL, pages 7567–7578.

Yue Weng, Sai Sumanth Miryala, Chandra Khatri, Runze Wang, Huaixiu Zheng, Piero Molino, M. Namazifar, A. Papangelis, H. Williams, Franziska Bell, and G. Tur. 2020. Joint contextual modeling for asr correction and language understanding. ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6349–6353.

Terry Winograd. 1971. Procedures as a representation for data in a computer program for understanding natural language. Technical report, Massachusetts Institute of Technology.

William Woods. 1972. The lunar sciences natural language information system. BBN report.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In ACL, pages 440–450.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In ACL.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In EMNLP, pages 3911–3921.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In Proceedings of the national conference on artificial intelligence, pages 1050–1055.

Luke S Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. arXiv preprint arXiv:1207.1420.

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

Su Zhu, Ouyu Lan, and Kai Yu. 2018. Robust spoken language understanding with unsupervised asr-error adaptation. In ICASSP, pages 6179–6183. IEEE.

A Method Details

A.1 Model Details

For the model structure diagrams, an LSTM block stands for a single LSTM layer. LSTM encoders are bidirectional and decoders are unidirectional. The pretrained LM, BERT or TaBERT, is frozen during training. Text-schema representations dimension is 768 (same as BERT-base); audio feature dimension is 68, including the values and delta values of 34 basic audio features; audio encoding (from 1D-CNN) dimension is 128; tag embedding dimension is 100; char-level embedding dimension is 128. LSTM hidden dimension is 256 (128 per direction for bidirectional LSTM). For Reranker, the FF layers are (Linear; Leaky-ReLU(0.01); Linear; Sigmoid). The middle dimension between FF layers is 64. The margin in the margin loss is 0.25. For all models, the optimizer is Adam with initial learning rate 0.001. Batch size is 8. We use early stopping with patience 10 epochs for Tagger, 50 epochs for ILM, based on preliminary study.

For text representation, we added a standard character embedding and a 1D-CNN char-level encoder to obtain the char-level representations of each token, concatenated with the PLM text representations.

We also attempted the following modifications in preliminary experiments (on ILM rewriter), but they did not improve the model performance: fine-tuning the PLM (all layers or top 2 layers); for audio attention, using other attention types (linear, bilinear, etc.); for encoder final encoding layer, using self-attention in place of LSTM; using more layers of audio-attention or LSTM; using SGD as the optimizer in place of Adam. For the Tagger, we are actually using a basic version (not using token phoneme losses), due to our findings in preliminary study that Tagger is not the performance bottleneck.

A.2 Pretrained LM

For BERT, we using BERT-base-uncased. For TaBERT, we use the version of K=1. Another detail is that TaBERT only encodes a query and a single table, instead of all tables in a DB. When using TaBERT as pretrained LM, we feed the text query and each table into TaBERT, obtaining t representation results where t is the number of tables. Each query token has t representations, which are averaged; each schema token only has one representation, which is directly used as its token representation.

A.3 Audio Encoder

Given the audio of a token, we apply a sliding window on the audio to compute a matrix of basic audio features[5], with shape `(num-of-windows, audio-feats-dim)`. We then apply a 1D-CNN on the time dimension, i.e. `num-of-windows`, to get the audio feature vector for the token, with length `audio-feats-dim`.

A.4 Fast Aligner

We use Fast Aligner to get the "gold rewriter tags". However, Fast Aligner is originally for SMT and it allows reordering of tokens during matching which should not occur in ASR tokens matching. Also, it only provides token-to-token matches. Therefore, we postprocess Fast Aligner output by merging successive tokens into spans and match spans with any token matches in them, until there is not more reordering. For all spans matches, if the text is identical, all tokens are tagged KEEP; if the text differ, all tokens are tagged EDIT (in BIOUL). Tokens without a match are tagged DEL.

A.5 Text-to-SQL parser

RAT-SQL Due to limited GPU memory we were unable to train RAT-SQL-BERT; instead, we used the Glove version. Our trained model achieved 62.4 exact match score on the dev set, close to the reported performance 62.7 on the leaderboard entry RATSQL v2 (DB content used).

Picard Again due to memory limit, we were unable to setup the Picard parsing module, so we used the officially released model version "t5.1.1.lm100k.large w/o Picard". It obtains 71.2 exact match on dev set, which is still much stronger than RAT-SQL-Glove.

B Experiment Details

B.1 Spoken Spider Statistics

The statistics of our Spoken Spider dataset are shown in Table 7. The dataset can be downloaded at https://drive.google.com/file/d/16w4E_W6BqMs0OvWWsFmD6YAziItNVJeX/view?usp=sharing.

[5]The basic audio features include frame energy, spectrum features, MFCC, etc. Features are extracted using pyAudioAnalysis library: https://github.com/tyiannak/pyAudioAnalysis

Dataset splits	# of clean queries	# of ASR candidate queries
Training	7000	41112
Dev	487	2707
Test	547	3075

Table 7: Spoken Spider statistics. The training set comes from "train_spider.json" in original Spider; the dev and test set come from "dev.json".

B.2 Partial Match Score

The partial match scores provided in Spider official evaluation scripts are the F1 scores of each type of clause (Select, Where, GroupBy, etc.) between predicted and gold SQL. To make it a single-value evaluation metric, we compute an average score across clause types, weighted by the occurrences of each type. This score is better than exact match score as the supervision signal for Reranker because it is continuous in range [0, 1] while exact match score is binary.

C Analysis Study Details

C.1 Oracle Analysis

Oracle Tagger Directly output the gold rewriter tags

Oracle ILM rewriter Based on the Fast Aligner alignment results (which we used to generate gold rewriter tags). For each `EDIT` span, rewrite the tokens with their aligned tokens in the gold utterance.

C.2 Syntactic Category Analysis (POS)

We use SpaCy to assign a POS tag to each token in the ASR transcription. We use a simple dynamic programming edit-distance algorithm to align the rewritten utterance to the gold one, check if each token correctly aligns with the rewritten query, and compute the percentage of tokens being correct for each POS tag respectively. This percentage is used as *token accuracy*.

The results are shown in Table 3 in main text. On the top of the table are the POS tags on which the ASR errors are relatively better addressed by the rewriter. For PUNCT, many ASR outputs have periods at the end when the queries are questions. Also, some ASR outputs have extra ending punctuation marks in the middle. Errors on NUM are usually formatting errors, such as numbers transcribed into English words or mismatches in comma delimiters. Some of these errors actually appeared in samples in Table 4 in main text. Generally, the errors types listed above are more patterned and the rewriter is able to handle them well. At the bottom is PROPN (proper nouns), which we have discussed in the main text.

For freezing or modifying experiments, we only used the run with medium performance (on exact match) among all 5 runs. We treat it as a representative of all runs.

C.3 Probe

For each probing task, the model of probes is logistic regressor. Probing F1 score is also obtained using the medium run among all 5 runs, similar to the freezing / modifying experiments in POS analysis.

For the utterance mentioning schema task and schema directly mentioned task (B, C), a mention is determined by word stem matching.

For the token phonemes task (E), the probing F1 is only averaged over phonemes with $P(pos) > 0.05$, because there are a lot of infrequent phonemes that have almost no positive labels in the dataset.

D Experiment Environment

CPU: 40 × Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz

GPU: 1 × Tesla P100-PCIE-12GB

CUDA: Version = 10.1

OS: Ubuntu 16.04.1 LTS (Xenial Xerus)