CQR-SQL: Conversational Question Reformulation Enhanced Context-Dependent Text-to-SQL Parsers

Dongling Xiao¹*, Linzheng Chai²*, Qian-Wen Zhang¹, Zhao Yan¹, Zhoujun Li² and Yunbo Cao¹

¹Tencent Cloud Xiaowei
²State Key Lab of Software Development Environment, Beihang University, Beijing, China

¹{dlxiaocowenzhang, zhaoyan, yunbocao}@tencent.com
²{challenging, lizj}@buaa.edu.cn

Abstract

Context-dependent text-to-SQL is the task of translating multi-turn questions into database-related SQL queries. Existing methods typically focus on making full use of history context or previously predicted SQL for currently SQL parsing, while neglecting to explicitly comprehend the schema and conversational dependency, such as co-reference, ellipsis and user focus change. In this paper, we propose CQR-SQL, which uses auxiliary Conversational Question Reformulation (CQR) learning to explicitly exploit schema and decouple contextual dependency for multi-turn SQL parsing. Specifically, we first present a schema enhanced recursive CQR method to produce domain-relevant self-contained questions. Secondly, we train CQR-SQL models to map the semantics of multi-turn questions and auxiliary self-contained questions into the same latent space through schema grounding consistency task and tree-structured SQL parsing consistency task, which enhances the abilities of SQL parsing by adequately contextual understanding. At the time of writing, our CQR-SQL achieves new state-of-the-art results on two context-dependent text-to-SQL benchmarks SPARC and CoSQL.

1 Introduction

The text-to-SQL task is one of the widely followed branches of semantic parsing, which aims to parse natural language questions with a given database into SQL queries. Previous works (Zhong et al., 2017; Yu et al., 2018; Wang et al., 2020) focus on context-independent text-to-SQL task. However, in reality, as users tend to prefer multiple turns interactive queries (Iyyer et al., 2017), the text-to-SQL task based on conversational context is attracting more and more scholarly attention. The generalization challenge of the context-dependent text-to-SQL task lies in jointly representing the multi-turn questions and database schema while considering the contextual dependency and schema structure.

As shown in Figure 1, to resolve the contextual dependency, the model should not only understand the co-reference and ellipsis, but also prevent from irrelevant information integration when user focus changes. Recent studies on two large-scale context-dependent datasets, SPARC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a), also show the difficulty of this problem. To our knowledge, there is a lack of explicit guidance for mainstream text-to-SQL researches dealing with contextual dependency.

For context-dependent text-to-SQL, it is common to train a model in an end-to-end manner that simply encoding the concatenation of the multi-turn questions and schema, as shown in Figure 2(a). To exploit context-dependence information, Hui et al. (2021) propose a dynamic relation decay mechanism to model the dynamic relationships between schema and question as conversation proceeds. Zhang et al. (2019) and Zheng et al. (2022) leverage previously predicted SQL queries to enhance currently SQL parsing. However, we argue that these end-to-end approaches are inadequate guidance for the contextual dependency phe-

Figure 1: An example of context-dependent Text-to-SQL task demonstrates the phenomenon of co-reference, ellipsis, and user focus changes. The CQR module converts contextual questions to self-contained questions, which can be understood without the context.
nomenon, though they are competitive in their evaluation of existing context modeling methods.

To help the models achieve adequate understanding of the current user question \( q_{\tau} \), conversational question reformulation (CQR) is crucial for multi-turn dialogue systems (Pan et al., 2019; Kim et al., 2021). As far as we know, only few works in contextual-dependent text-to-SQL, such as (Chen et al., 2021), focus on the value of CQR for modeling question context. Chen et al. (2021) propose a two-stage pipeline method in which an CQR model first generates a self-contained question \( \tilde{r}_{\tau} \), and then a context-independent text-to-SQL parser follows, as shown in Figure 2(b). But in practice, the limitations of the two-stage pipeline method are in two aspects: 1) the error propagation from the potentially wrong \( \tilde{r}_{\tau} \) to the single-turn text-to-SQL parser; 2) the neglect of the relevance between the two stages. Besides, CQR for text-to-SQL is more challenging than the general CQR tasks (Pan et al., 2019; Elgohary et al., 2019), since multi-turn questions in text-to-SQL datasets are strictly centered around the underlying database and there are no CQR annotations on existing text-to-SQL datasets.

Motivated by these observations, we propose CQR-SQL, which uses auxiliary CQR to achieve adequately contextual understanding, without suffering from the limitations of two-stage methods. Accordingly, we first introduce an schema enhanced recursive CQR method to product self-contained question data, as in “Stage 1” of Figure 2(c). The design not only integrates the underlying database schema \( D \), but also inherits previous self-contained question \( \tilde{r}_{\tau-1} \) to improve the long-range dependency. Secondly, we propose to train model mapping the self-contained questions and the multi-turn question context into the same latent space through schema grounding consistency task and tree-structured SQL parsing consistency task, as in Figure 2(d1). In this way, to make similar prediction as self-contained question input, models need to pay more attention to the co-reference and ellipsis when encoding the question context. As shown in Figure 2(d2), during inference, CQR-SQL no longer relies on the self-contained questions from CQR models, thus circumventing the error propagation issue of two-stage pipeline methods.

We evaluated CQR-SQL on SPARC and CoSQL datasets, and our main contributions of this work are summarized as follows:

- We present a schema enhanced recursive CQR mechanism that steadily generates self-contained questions for context-dependent text-to-SQL.
- We propose two novel consistency training tasks to achieve adequate contextual understanding for context-dependent SQL parsing by leveraging auxiliary CQR, which circumvents the limitations of two-stage pipeline approaches.
- Experimental results show that CQR-SQL achieves state-of-the-art results on context-dependent text-to-SQL benchmarks, SPARC and CoSQL, with abilities of adequate context understanding.

2 Proposed Method

In this section, we first formally define the context-dependent text-to-SQL task and introduce the backbone network of CQR-SQL. Afterwards, the technical details of CQR-SQL are elaborated in two subsections: Schema enhanced recursive CQR and Latent CQR learning for text-to-SQL in context.

2.1 Preliminary

Task Formulation. In context-dependent text-to-SQL tasks, we are given multi-turn user questions \( q = \{q_1, q_2, ..., q_n\} \) and the schema \( D = \langle T, C \rangle \) of target database which contains a set of tables \( T = \{t_1, t_2, ..., t_T\} \) and columns \( C_i = \{c_{i1}, c_{i2}, ..., c_{i|C_i|}\} \) for the \( i \)-th table \( t_i \). Our goal is to generate the target SQL query \( s_{\tau} \) with the question context \( q_{\leq \tau} \) and schema information \( D \) at each question turn \( \tau \).

Backbone Network. CQR-SQL takes multi-turn questions \( q \) as input along with the underlying
database schema $D$ in the Encoder-Decoder framework. For encoder, CQR-SQL employs the widely used relation-aware Transformer (RAT) encoder (Wang et al., 2020) to jointly represent question and structured schema. For decoder, CQR-SQL follows the tree-structured LSTM of Yin and Neubig (2017) to predict the grammar rule of SQL abstract syntax tree (AST), column $\vdash d$ and table $\vdash d$ at each decoding step, indicated as APPLYRULE, SELECTCOLUMN and SELECTTABLE (See Appendix A for detailed descriptions).

### 2.2 Schema Enhanced Recursive CQR

Due to the scarcity of in-domain CQR annotations for context-dependent text-to-SQL, we adopt self-training with schema enhanced recursive CQR method to collect reliable self-contained questions.

#### Schema Integration for CQR. 

Multi-turn questions in text-to-SQL are centered around the underlying database. To generate more domain relevant self-contained question $r_\tau$ at each turn $\tau$, we concatenate the question context $q_{\leq \tau}$ with schema $D$ as input $x_\tau = \{q_1, \text{[SEP]}, \ldots, q_\tau, \text{[SEP]}, t_1, c_{11}, c_{12}, \ldots, \text{[SEP]}, t_2, c_{21}, c_{22}, \ldots\}$. The BoW loss of $S_G$ is formulated as:

$$L_{cqr}^{\text{BoW}} = -\log \mathcal{P}(r_\tau | \{ \hat{r}_{\tau-1}, \text{[SEP]}, x_\tau \}).$$  

(1)

During training, other than using the labeled self-contained questions $r_{\tau-1}$ as $\hat{r}_{\tau-1}$, we sampled $\tilde{r}_{\tau-1}$ from a pre-trained CQR model to reduce discrepancies between training and inference.

#### Self-training for CQR.

Chen et al. (2021) indicate that models trained with general CQR datasets work poor on the in-domain data from CoSQL and SPARC. Besides the annotated in-domain self-contained question data is scarce for all context-dependent text-to-SQL tasks.

We conduct a self-training approach with a pre-trained single-turn text-to-SQL model $\theta_{\text{SQL}}$ to collect full self-contained question data $D^{\text{cqr}}$ for text-to-SQL datasets $D$, as shown in Algorithm 1.

### 2.3 CQR-SQL : Latent CQR Learning for Text-to-SQL Parsing in Context

With the self-contained questions $D^{\text{cqr}}$ in §2.2, during training, we introduce CQR-SQL, which uses a latent variable [2] to map the semantics of question context and self-contained question into the same latent space with two consistency tasks (schema grounding and SQL parsing), helping models achieve adequately contextual understanding for enhanced SQL parsing during inference.

As shown in Figure 3(a), during training, we input $\text{Seq}(q) = \{ 1, 2, \text{[SEP]}, D \}$ to CQR-SQL, where $q$ can be the question context $q_{\leq \tau}$ or self-contained questions $r_{\tau}$.

#### Schema Grounding Consistency Task. 

Grounding tables and columns into question context requires adequately understanding the co-reference and ellipsis in multi-turn questions. Thus we propose using the hidden state $z$ of latent variable to predict the tables and columns appear in current target SQL query $s_\tau$ with bag-of-word (BoW) loss (Zhao et al., 2017), and then enforcing models to make consistent predictions with question context input and self-contained question input, as shown in Figure 3(a). The BoW loss of Schema Grounding task $L_{\text{BoW}}^{SG}$ at each turn $\tau$ are formulated as:

$$L_{\tau}^{\text{SG-BoW}} = \text{BoW}(q_{\leq \tau}) + \text{BoW}(r_{\tau}).$$  

(2)

$$\text{BoW}(q) = -\log \mathcal{P}(D_{\tau} | \text{Seq}(q)) = -\sum_{d \in D_{\tau}} \sum_{d \in D_\tau} e^{f_d(D_{\tau} | \text{Seq}(q))}. $$  

(3)

where $\hat{D}_{\tau}$ refers to the schema appeared in current SQL query $s_\tau$, $D$ indicates the full schema of target database. $\mathcal{P}(\hat{D}_{\tau})$ represents the schema prediction probability distributions at turn $\tau$. The function $f_d(z) = h_d \mathbf{W}_{\text{SQ}} z$. $h_d$ denotes the final hidden states of schema $d$ for RAT encoder.
z_0 = RAT(\( \text{Seq}(q_{<\tau}) \)) and z_\tau = RAT(\( \text{Seq}(r_\tau) \)) to indicate the final hidden state of the latent variables associated with question context \( q_{<\tau} \) and self-contained question \( r_\tau \) respectively. The Schema Grounding consistency loss \( L_{SGKL}^{\tau} \) is defined as:

\[
L_{SGKL}^{\tau} = KL \left( \mathbb{P}(\hat{D}_\tau | \text{Seq}(q_{<\tau})) \| \mathbb{P}(\hat{D}_\tau | \text{Seq}(r_\tau)) \right) \\
+ KL \left( \mathbb{P}(\hat{D}_\tau | \text{Seq}(q_{<\tau})) \| \mathbb{P}(\hat{D}_\tau | \text{Seq}(q_{<\tau})) \right),
\]

where \( KL(\cdot) \) refers to the Kullback–Leibler divergence between two distributions.

**SQL Parsing Consistency Task.** Furthermore, to encourage model pay more attention to the SQL logic involving co-reference and ellipsis, we introduce to enforce the model to obtain the consistency prediction of SQL parsing with question contexts and self-contained questions as inputs, at each decoding step. The SQL parsing loss \( L_{SP}^{\tau} \) and the SQL Parsing consistency loss \( L_{SPKL}^{\tau} \), at each turn \( \tau \), can be represented as:

\[
L_{SP}^{\tau} = -\log \mathbb{P}(s_\tau | \text{Seq}(q_{<\tau})) - \log \mathbb{P}(s_\tau | \text{Seq}(r_\tau))
\]

\[
L_{SPKL}^{\tau} = KL \left( \mathbb{P}(s_\tau | \text{Seq}(q_{<\tau})) \| \mathbb{P}(s_\tau | \text{Seq}(r_\tau)) \right) \\
+ KL \left( \mathbb{P}(s_\tau | \text{Seq}(r_\tau)) \| \mathbb{P}(s_\tau | \text{Seq}(q_{<\tau})) \right).
\]

In this work, we follow the tree-structured decoder of Yin and Neubig (2017), which generates SQL queries as an abstract syntax tree (AST), and conduct three main predictions at each decoding step, including \( \text{APPLYRULE} \), \( \text{SELECTCOLUMN} \), and \( \text{SELECTTABLE} \). We calculate the SQL parsing consistency loss by accumulating all KL divergences of above three predictions as \( KL(\cdot) = KL_{\text{APPLYRULE}}(\cdot) + KL_{\text{SELECTCOLUMN}}(\cdot) + KL_{\text{SELECTTABLE}}(\cdot) \) at all decoding steps, as shown in Figure 3(b) and further described in Appendix A.3.

Finally we calculate the total training loss \( L_\tau \) at each question turn \( \tau \) for our context-dependent text-to-SQL model CQR-SQL as:

\[
L_\tau = L_{SP}^{\tau} + \lambda_1 L_{SGKL}^{\tau} + \lambda_2 \left( KL_{SPKL}^{\tau} + KL_{SGKL}^{\tau} \right)
\]

where \( \lambda_1 \) and \( \lambda_2 \) are weights for the schema grounding BoW loss and the consistency loss respectively.

**CQR-SQL Inference.** Since CQR-SQL has learned to adequately understand the context dependency in question context \( q_{<\tau} \) by distilling representations from self-contained question in two consistency tasks, CQR-SQL no longer relies on self-contained questions and only considers \( \text{Seq}(q_{<\tau}) \) as inputs, as shown in Figure 2(d2), thus circumventing the error propagation in two-stage pipeline methods.

### 3 Experiments

In this section, we conduct several experiments to assess the performance of proposed methods in §2.

#### 3.1 Experimental Setup

**CQR Learning.** We adopt the Transformer-based encoder-decoder architecture based on the pretrained ProphetNet (Qi et al., 2020) as the initial CQR model. Since there is no question reformulation annotations in SPARC and CoSQL, we annotate 3034 and 1527 user questions as the initial in-domain supervised CQR data \( D_{0}^{\text{CQR}} \) for SPARC and CoSQL respectively. Before self-training, we pre-train a single-turn text-to-SQL model \( \theta_{\text{SQL}} \) based on RAT-SQL (Wang et al., 2020) architecture and ELECTRA (Clark et al., 2020) language.
Table 1: Performances on the development and test set of SPARC and CoSQL. “QM” and “IM” indicate the exact match accuracy over all questions and all interaction respectively. The models with ♦ mark employ task adaptive pre-trained language models. Models with ♢ mark use the general two-stage pipeline approach in Figure 2(b). The “−” results of CQR-SQL are awaiting evaluation due to the submission interval of the leaderboard.

Table 2: Detailed statistics for SPARC and CoSQL.

model for checking whether a generated question is self-contained enough for correctly SQL parsing. During self-training in §2.2, we conduct 3 training loops \{θ₁, θ₂, θ₃\} and obtain 4441 and 1973 supervised CQR data for SPARC and CoSQL respectively. Finally, we use the CQR model θ₃ in the last training loop to produce the self-contained questions for all interaction turns.

CQR-SQL Training. We conduct experiments on two context-dependent text-to-SQL datasets SPARC and CoSQL, the statistic information of them are depicted in Table 2. Following Cao et al. (2021), we employ RAT-SQL (Wang et al., 2020) architecture and pre-trained ELECTRA (Clark et al., 2020) for all text-to-SQL experiments in this paper. In the training of CQR-SQL, we set hyperparameters λ₁ = 0.1 and λ₂ = 3.0 for SPARC, λ₂ = 1.0 for CoSQL (See Appendix B.2 for details), learning rate as 5e-5, batch size of 32. During inference, we set the beam size to 5 for SQL parsing.

3.2 Experimental Results

As shown in Table 1, CQR-SQL achieves state-of-the-art results cross all settings at the time of writing. With general PLM BERT, CQR-SQL surpasses all previous methods, including the two-stage method DELTA (Chen et al., 2021) which also uses additional text-to-SQL data from Spider. Beside, most of recent advanced methods tend to incorporates more task-adaptive data (text-table pairs and synthesized text-sql pairs), tailored pre-training tasks (column prediction and turn switch prediction) and super-large PLM T5-3B (Raffel et al., 2020) into training. For this setting, we use general PLM ELECTRA for all text-to-SQL experiments following Cao et al. (2021), and further employ a more compatible¹ PLM COCO-LM (Meng et al., 2021) for comparison. CQR-SQL significantly outperforms SCORE (Yu et al., 2021b), RAT-SQL+TC (Li et al., 2021) and recent HIE-SQL (Zheng et al., 2022) which use task-adaptive pre-trained models. Note that HIE-SQL employs two task-adaptive PLMs for encoding text-schema pairs and previous SQL queries respectively. Compared with methods based on super-large T5-3B model (especially RASAT (Qi et al., 2022) which integrates co-reference relations and constrained decoding into T5-3B), CQR-SQL can also achieve significant improvements.

To verify the advantages of CQR-SQL on adequately contextual understanding, we further compare the performances on different interaction turns of SPARC, as shown in Figure 4(a). We observe

1 COCO-LM is pre-trained on sequence contrastive learning with a dual-encoder architecture (Reimers and Gurevych, 2019), which is compatible for our CQR consistency tasks with dual-encoder for multi-turn q_c and self-contained r_c.
that it is more difficult for SQL parsing in longer interaction turns due to the long-range dependency problem, while CQR-SQL achieves more significant improvement as the interaction turn increases. Moreover, in Figure 4(b), we further compare the performances on varying difficulty levels of target SQL queries. CQR-SQL consistently outperforms previous works on all difficulty levels, especially on the “Extra Hard” level whose target SQL queries are most complex and usually contain nesting SQL structures (Yu et al., 2018).

3.3 Ablation Study

Regarding the CQR task, as shown in Table 3, recursive generation (RG) achieves 0.46% BLEU score gains on the CQR task for CoSQL dataset which has much longer interaction turns than SPARC as shown in Table 2, while RG fails to significantly improve the performance for SPARC_CQR. This indicates RG can improve CQR performance for longer contextual dependency. While further removing the schema enhanced SE method, performances decrease by roughly 1% and 0.5% on SPARCCQR and CoSQLCQR respectively, which verifies the effectiveness of schema integration in CQR for text-to-SQL datasets. We additionally evaluate the performances without any in-domain CQR annotations (fine-tune T5-3B on general CQR dataset CANARD (Elgohary et al., 2019)), and observe that performances on CQR tasks are more significantly reduced than on CoSQL, verifying the effect of in-domain data and the robustness of CQR-SQL against noised CQR information.

In Table 4, we investigate the contribution of each designed choice of proposed CQR-SQL.

Table 3: Comparisons on the BLEU / Rouge-L scores between scheme enhanced (SE) approach and recursive generation (RG) for CQR. Models are trained and evaluated on the initially annotated data D_{SE}^{tr}. We observe that the BLEU / Rouge-L scores of CoSQL_CQR are much higher than those of SPARC, because CoSQL has much more user focus change questions that without co-references and ellipses (Yu et al., 2019a).

Table 4: Ablation studies for CQR-SQL and its variants. SG denotes the Schema Grounding task (including BoW loss and consistency loss SG_{KL}), and SP_{KL} denotes the SQL Parsing consistency task.

Figure 4: Detailed question match (QM) accuracy results in different interaction turns and goal difficulties on the dev set of SPARC dataset. # Number denotes the number of questions. Detailed results of c (Li et al., 2021),b (Hui et al., 2021),c (Cai and Wan, 2020) and d (Zhang et al., 2019) are from the original paper.
List the name and date of the battle that has lost the ship named ‘HMS Atlanta’ and ‘Lettice’.

What about those who do not have any dogs temporarily?

Two-Stage: SELECT battle.name, battle.date FROM battle JOIN ship WHERE ship.name = "value"^

CQR-SQL: SELECT battle.name, battle.date FROM battle JOIN ship WHERE ship.name = "value"^

End-to-end: SELECT cars_data.Model FROM cars_data WHERE cars_data.Horsepower = "value"^

CQR-SQL: SELECT cars_data.Model FROM cars_data WHERE cars_data.Horsepower = "value"^

Figure 5: Case studies on SPARC dataset. Upper block shows the cases of error propagation with incorrectly generated self-contained questions $\tau_1$ for Two-Stage pipeline methods (as in Figure 2(c) or [7] in Table 3). Cases in the lower block show that End-to-End method (as in Figure 2(a) or [4] in Table 4) fails to resolve the conversational dependency. Besides, the heat maps represent the visualization of attention scores of the latent variable $\tau_1$ on the question context part. Number on each dotted box is the average attention scores of that box.

[6]: CQR_Augment. We directly augment the full self-contained question data $D^{\text{qp}}$ to [4] as single-turn text-to-SQL data augmentation. We find CQR_Augment degrades the performances because models are trained more on single-turn text-to-SQL, while weaken the abilities of contextual understanding for multi-turn text-to-SQL.

[7]: CQR_TwoStage*. A variant trained in the improved “Two-Stage” manner as in Figure 2(c). It slightly outperforms the End-to-End variant [4].

[8]: CQR_TwoStage. A “Two-Stage” variant without additionally integrating question context into “Stage 2” as variant [7]. The performances significantly decline for the error propagation issue.

[9]: CQR_MultiTask. A variant jointly trained on CQR task and text-to-SQL task with a shared RAT encoder and two task-specific decoders, as shown in Figure 6. We observe that CQR_MultiTask slightly decreases the performances compared with End-to-End variant [4] for the optimization gap between the text-to-text optimization and structured text-to-SQL optimization.

Figure 6: Schematic of CQR_MultiTask variant.

Above results and analyses demonstrate the advantages of CQR-SQL on leveraging self-contained questions to enhance the abilities of adequately context understanding for contextual text-to-SQL parsing, meanwhile circumventing the error propagation of Two-Stage pipeline methods.

3.4 Case Study

As shown in the upper block of Figure 5, we compare the SQL queries by CQR-SQL with those by the Two-Stage baseline model in Figure 2(c) or [7] in Table 3, to show the error propagation phenomenon between CQR stage and text-to-SQL stage. The generated self-contained questions $\tau_1$ are incorrect, leading to wrong predictions of SQL queries. Specifically, In the first case, CQR model fails to understand “intersect” in current question $q_3$, thus missing “Lettice” in the generated self-contained question $\tau_3$ and leading to uncompleted SQL queries in the text-to-SQL stage. In the second case, CQR model misunderstood “combine” in the the current question $q_3$, leading to incorrect key word “AND” in the predicted SQL query.

In the lower block of Figure 5, we compare the SQL queries by CQR-SQL with those by the End-to-End baseline, and visualize the attention patterns of latent variable $\tau_2$ on question context. The first case shows the scenario which requires model to inherit history information to resolve the co-reference and ellipsis. From the heat map, we observe that latent variable $\tau_2$ pays more attention to the desirable history information “volvo” and “least accelerate”. While the second case shows the scenario which requires model to discard confusion history
information. In this case, latent variable $[Z]$ pays less attention to the confusion information block "state of Arizona" compared with the desirable ones, which is benefit for correctly SQL parsing.

### 3.5 Transferability on Contextual Text-to-SQL

To verify the transferability of CQR integration on contextual text-to-SQL task, we conduct three out-of-distribution experiments as shown in Table 5. In experiment [1], our CQR-SQL has better transferability on contextualized questions (Turn $\geq 2$) in CoSQL$_{Dev}$, which contains additional system response and more question turns compared with SPARC dataset as in Table 2. Beside, in experiment [2–3], CQR-SQL achieves consistently better performances on out-of-distribution contextual questions (Turn $\geq 3$). These results indicate the advantage of CQR-SQL in robustness contextual understanding for out-of-distribution scenarios.

Table 5: Question match (QM) accuracy results of the out-of-distribution experiments. $S$ and $C$ denote SPARC and CoSQL datasets respectively. **Experiment [1]**: training models on the training set of SPARC$_{Train}$, while evaluating them on the questions with context (Turn $\geq 2$) in CoSQL$_{Dev}$. **Experiment [2–3]**: training models on the questions at Turn $\leq 2$, whereas evaluating them on the questions at Turn $\geq 3$.

<table>
<thead>
<tr>
<th>#</th>
<th>Train</th>
<th>Eval</th>
<th>End-to-End</th>
<th>Two-Stage</th>
<th>CQR-SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>$S_{Train}$</td>
<td>$C_{Train}$</td>
<td>$C_{Dev}$</td>
<td>$C_{Dev}$</td>
<td>$C_{Dev}$</td>
</tr>
<tr>
<td>[2]</td>
<td>$S_{Train}$</td>
<td>$C_{Turn=2}$</td>
<td>$C_{Dev}$</td>
<td>$C_{Dev}$</td>
<td>$C_{Dev}$</td>
</tr>
<tr>
<td>[3]</td>
<td>$C_{Train}$</td>
<td>$C_{Turn=2}$</td>
<td>$C_{Dev}$</td>
<td>$C_{Dev}$</td>
<td>$C_{Dev}$</td>
</tr>
</tbody>
</table>

### 4 Related Work

**Text-to-SQL.** Spider (Yu et al., 2018) is a well-known cross-domain context-independent text-to-SQL task that has attracted considerable attention. Diverse approaches, such as RAT-SQL (Wang et al., 2020), BRI$\text{DGE}$ (Lin et al., 2020) and LGESQL (Cao et al., 2021), have been successful on this task. Recently, with the widespread popularity of dialogue systems and the public availability of SPARC (Yu et al., 2019b) and CoSQL (Yu et al., 2019a) datasets, context-dependent text-to-SQL has drawn more attention. Zhang et al. (2019) and Wang et al. (2021) use previously generated SQL queries to improve the quality of SQL parsing. Cai and Wan (2020) and Hui et al. (2021) employ graph neural network to model contextual questions and schema. Jain and Lapata (2021) use a memory matrix to keep track of contextual information. Chen et al. (2021) decouple context-dependent text-to-SQL task to CQR and context-independent text-to-SQL tasks. Besides, Yu et al. (2021a,b) propose task-adaptive conversational pre-trained model for SQL parsing, and Scholak et al. (2021) simply constrain auto-regressive decoders of super-large T5-3B for SQL parsing. In this work, we leverage reformulated self-contained questions in two consistency tasks to enhance contextual dependency understanding for multi-turn text-to-SQL parsing, without suffering from the error propagation of two-stage pipeline methods.

**Conversational Question Reformulation (CQR)** aims to use question context to complete ellipsis and co-references in the current questions. Most works adopt the encoder-decoder architecture with only contextual text as input (Elgohary et al., 2019; Pan et al., 2019). Besides, CQR are applied to several downstream tasks for enhanced context understanding, such as conversational question answer (CQA) (Kim et al., 2021) and conversational passage retrieval (CPR) (Dalton et al., 2020). Regarding CQR training for text-to-SQL, We present a recursive CQR method to address long-range dependency and incorporate schema to generate more domain-relevant and semantic-reliable self-contained questions.

**Consistency Training.** To improve model robustness, consistency training (Zheng et al., 2016) has been widely explored in natural language processing by regularizing model predictions to be invariant to small perturbations. The small perturbations can be random or adversarial noise (Miyato et al., 2018) and data augmentation (Zheng et al., 2021). Inspired by consistency training, ExCorD (Kim et al., 2021) trains a classification CQA model that encourage the models to predict similar answers span from the rewritten and original questions. Different from ExCorD, we combine latent variable with schema grounding consistency task and tree-structured SQL generation consistency task to force model pay more attention to the co-references and ellipsis in question context.

### 5 Conclusions

We propose CQR-SQL, a novel context-dependent text-to-SQL approach that explicitly comprehends the schema and conversational dependency through latent CQR learning. The method introduces a schema enhanced recursive generation mechanism to generate domain-relative self-contained questions, then trains models to map the semantics of self-contained questions and multi-turn question
context into the same latent space with schema grounding consistency task and SQL parsing consistency task for adequately context understanding. Experimental results show that CQR-SQL achieves new state-of-the-art results on two classical context-dependent text-to-SQL datasets SPARC and CoSQL.

6 Limitations

Compared to End-to-End approaches as shown in Figure 2(a), proposed CQR-SQL requires more computational cost and GPU memory consumption at each training step, with duel-encoder for question context and self-contained question inputs. Specifically, with batchsize of 32 and 8 V100 GPU cards, CQR-SQL takes 310.7 seconds to train an epoch on SPARC dataset, while End-to-End approach costs 179.6 seconds. While compared with previous advanced methods using T5-3B PLM (Scholak et al., 2021; Xie et al., 2022; Qi et al., 2022), and multiple task-adaptive PLMs (Zheng et al., 2022), CQR-SQL is much more computationally efficient.

Acknowledgments

We are indebted to the EMNLP2022 reviewers for their detailed and insightful comments on our work. This work was supported in part by the National Natural Science Foundation of China (Grant No. 62276017), and the 2022 Tencent Big Travel Rhino-Bird Special Research Program.

References


A Backbone Architecture

A.1 Encoder: Relation-aware Transformer

Relation-aware Transformer (RAT) (Wang et al., 2020) is an extension to Transformer (Vaswani et al., 2017) to consider preexisting pairwise relational features between the inputs. For text-to-SQL task, pairwise relational features include the intra-relation of database schema D and question-schema alignment information. Formally, given input sequence \(x = \{x_1, x_2, ..., x_n\}\), we obtain the initial representations through pre-trained language model \(H^0 = PLM(x)\) = \(\{x_1, x_2, ..., x_n\}\). Then \(L\) stacked RAT blocks compute the final hidden states \(H^L = RAT(x)\) via \(H^l = RAT_l(H^{l-1})\), \(l \in [1, L]\) where \(RAT_l(\cdot)\) is calculated as:

\[
Q = H^{l-1}W_Q, K = H^{l-1}W_K, V = H^{l-1}W_V
\]

\[
e_{ij} = \frac{\text{Softmax}_{n,j} \left( \frac{Q_i(K_j + r_{ij}^V)}{\sqrt{d_k}} \right)}{a_{ij}}
\]

\[
A = \text{LayerNorm}(H^{l-1} + A)
\]

\[
H^l = \text{LayerNorm} \left( \tilde{A} + \text{FC}(\text{ReLU}(\text{FC}(\tilde{A}))) \right)
\]

where parameters \(W_Q, W_K, W_V \in \mathbb{R}^{d_h \times d_k}\) project \(H^l\) to queries, keys and values. Embedding \(r_{ij}\) represents the relationship between token \(x_i\) and \(x_j\). LayerNorm(\(\cdot\)) is the layer normalization (Ba et al., 2016), FC(\(\cdot\)) is the full connected layer.

In CQR-SQL, the relation type between latent variable \([2]\) and schema is NO MATCH (Wang et al., 2020).

A.2 Decoder: Tree-structured LSTM

We employ a single layer tree-structured LSTM decoder of Yin and Neubig (2017) to generate the abstract syntax tree (AST) of SQL queries in depth-first, left-to-right order. At each decoding step, the prediction is either 1) \(\text{APPLYRULE}\) action that expands the last non-terminal into a AST grammar rule; or 2) \(\text{SELECTCOLUMN}\) or \(\text{SELECTTABLE}\) action that chooses a column or table from schema to complete last terminal node.

Formally, given the final encoder hidden states of Question, Table and Column \(H^T = \{H_Q, H_T, H_C\}\). The tree-structured decoder is required to generate a sequence of actions \(a_t\) to construct the AST which can transfer to standard SQL query \(s\), represented as \(P(s|H^T) = \prod_t P(a_t|a_{<t}, H^T)\). We adopt a single layer LSTM to produce action sequence, the LSTM states are updated as \(c_t, h_t = \text{LSTM}[a_{t-1}; h_{t-1}; n_f_t; n_s_t, c_{t-1}, h_{t-1}]\) where \([\cdot]\) is the concatenate operation, \(c_t\) is the LSTM cell state, \(h_t\) is the LSTM output hidden state, \(a_t\) is the action embedding, \(n_f_t\) is the decoding step of the current parent AST node and \(n_s_t\) is the embedding of current node type. We initialize the LSTM hidden state \(h_0\) via attention pooling over the final encoder hidden state \(H^L\) as:

\[
e_i = \text{Softmax}_1 \left( \hat{h}_0 \text{Tanh}(H^l_i W_1) \right)
\]

\[
A = \sum_{i=1}^n e_i h_i^l
\]

\[
h_0 = \text{Tanh}(A W_2).
\]

where \(\hat{h}_0\) is initial attention vector, \(W_1, W_2\) are projection parameters. At each decoding step \(t\), we employ MultiHeadAttention (Vaswani et al., 2017) on \(h_0\) over \(H^T\) to compute context representation \(h_i^{tx}\).

For the prediction of \(\text{APPLYRULE}\) actions, the prediction distribution is computed as:

\[
P(a_t = \text{APPLYRULE}[R]|a_{<t}, H^L) = \text{Softmax}_1(\text{MLP}_2([h_t; h_i^{tx}]) W_R).
\]

For the prediction of \(\text{SELECTTABLE}\) actions, the prediction distribution is computed as:

\[
P(a_t = \text{SELECTTABLE}[i]|a_{<t}, H^L) = \text{Softmax}_1(\text{MLP}_2([h_t; h_i^{tx}]) W_i).
\]

where \(h_{t_i}\) denotes the encoder hidden state of the \(i\)-th Table. The prediction of \(\text{SELECTCOLUMN}\) actions is similar to \(\text{SELECTTABLE}\).

A.3 SQL Parsing Consistency Loss

Given the final encoder hidden states \(H^L_T = \text{RAT}(\text{Seq}(q_{<T}))\) and \(H^L_Q = \text{RAT}(\text{Seq}(q_{<T}))\) for question context and self-contained question input respectively, the SQL parsing consistency loss \(L^{\text{SP}_{\text{KL}}}(t)\) at each decoding step \(t\) is computed as:

\[
L^{\text{SP}_{\text{KL}}}(t) = L^{\text{SP}_{\text{KL}}}_{\text{APPLYRULE}}(t) + L^{\text{SP}_{\text{KL}}}_{\text{SELECTTABLE}}(t)
\]

\[
+ L^{\text{SP}_{\text{KL}}}_{\text{SELECTCOLUMN}}(t)
\]

\[
L^{\text{SP}_{\text{KL}}}_{\text{APPLYRULE}} = \text{KL}(P(a_t = \text{APPLYRULE}|a_{<t}, H^L_T) || P(a_t = \text{APPLYRULE}|a_{<t}, H^L_Q)).
\]

\[
L^{\text{SP}_{\text{KL}}}_{\text{SELECTTABLE}} = \text{KL}(P(a_t = \text{SELECTTABLE}|a_{<t}, H^L_T) || P(a_t = \text{SELECTTABLE}|a_{<t}, H^L_Q)).
\]

The total SQL parsing consistency loss is computed as \(L^{\text{SP}_{\text{KL}}} = \sum_t L^{\text{SP}_{\text{KL}}}(t)\), where \(T\) denotes the length of action sequence for target SQL AST.
B Experiment Details

B.1 Detailed Hyperparameters

We implement C QR-SQL based on the PyTorch framework and use 8 Nvidia Tesla V100 32GB GPU cards for all the experiments. Firstly, we trained CQR model with a learning rate of 3e-5 and batch size of 32. We use the maximum input sequence length as 512 and the maximum epochs as 25. We adopt label smooth method with ratio 0.15 for regularization. During inference for CQR, we set the beam size as 5.

Regarding training CQR-SQL, the number of heads is 8 and hidden size of RAT encoder is 1024, the dropout rates of encoder and decoder are 0.1 and 0.2 respectively. For pre-trained ELECTRA, we adopt layer-wise learning rate decay with coefficient 0.8 for robust optimization. We train CQR-SQL on SPARC and CoSQL with max training epochs to be 300 and 350 respectively.

B.2 Impact of Weight $\lambda_2$ for Consistency Loss

We vary the weight $\lambda_2$ of consistency loss in $\{1.0, 2.0, 3.0, 4.0\}$ and train CQR-SQL on SPARC and CoSQL datasets, as shown in Table 6. We observe that CoSQL task desires less CQR knowledge (best choice of $\lambda_2$ is 1.0) compared with SPARC (best choice of $\lambda_2$ is 3.0), because CoSQL dataset contains much more user focus change questions than SPARC, which do not need to be reformulated (Yu et al., 2019b).

<table>
<thead>
<tr>
<th>Weight $\lambda_2$</th>
<th>SPARC$_{Dev}$ QM</th>
<th>SPARC$_{Dev}$ IM</th>
<th>CoSQL$_{Dev}$ QM</th>
<th>CoSQL$_{Dev}$ IM</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_2 = 1.0$</td>
<td>66.5</td>
<td>47.2</td>
<td>58.2</td>
<td>29.4</td>
</tr>
<tr>
<td>$\lambda_2 = 2.0$</td>
<td>67.1</td>
<td>47.6</td>
<td>58.0</td>
<td>28.3</td>
</tr>
<tr>
<td>$\lambda_2 = 3.0$</td>
<td>67.8</td>
<td>48.1</td>
<td>57.4</td>
<td>27.3</td>
</tr>
<tr>
<td>$\lambda_2 = 4.0$</td>
<td>66.0</td>
<td>46.7</td>
<td>56.7</td>
<td>26.6</td>
</tr>
</tbody>
</table>

Table 6: Results of CQR-SQL on SPARC and CoSQL datasets with different weights $\lambda_2$ of consistency loss.

B.3 Detailed Results on CoSQL Task

As shown in Table 7, we report the detailed results in different question turns and SQL difficulty levels on the development set of CoSQL dataset. We observe that CQR-SQL achieves more significant improvement as the interaction turn increases, and consistently outperforms previous works on all SQL difficulty levels.

<table>
<thead>
<tr>
<th>Models</th>
<th>Turn 1</th>
<th>Turn 2</th>
<th>Turn 3</th>
<th>Turn 4</th>
<th>Turn &gt;4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># 293</td>
<td># 285</td>
<td># 244</td>
<td># 114</td>
<td># 71</td>
</tr>
<tr>
<td>EditSQL$^a$</td>
<td>50.0</td>
<td>36.7</td>
<td>34.8</td>
<td>43.0</td>
<td>23.9</td>
</tr>
<tr>
<td>IGSQ$^b$</td>
<td>53.1</td>
<td>42.6</td>
<td>39.3</td>
<td>43.0</td>
<td>31.0</td>
</tr>
<tr>
<td>IST-SQL$^c$</td>
<td>56.2</td>
<td>41.0</td>
<td>41.0</td>
<td>41.2</td>
<td>26.8</td>
</tr>
<tr>
<td>SCORe$^d$</td>
<td>60.8</td>
<td>53.0</td>
<td>47.5</td>
<td>49.1</td>
<td>32.4</td>
</tr>
<tr>
<td>CQR-SQL</td>
<td>66.2</td>
<td>60.0</td>
<td>54.5</td>
<td>54.4</td>
<td>39.4</td>
</tr>
</tbody>
</table>

Table 7: Detailed QM results in different interaction turns and goal difficulties on the development set of CoSQL dataset. Detailed results of $^a$ (Zhang et al., 2019),$^b$ (Cai and Wan, 2020),$^c$ (Wang et al., 2021) and $^d$ (Yu et al., 2021b) are from the original paper.

B.4 Effects of CQR Integration with Different PLMs

To further study the effects of CQR integration for contextual text-to-SQL task, we train models in End-to-End, Two-Stage and CQR-SQL approaches based on different pre-trained language models (PLMs), as shown in Table 8. We can see that: 1) CQR-SQL method consistently performs better than Two-Stage and End-to-End methods, further demonstrating the effectiveness of CQR-SQL for adequate contextual understanding. 2) COCO-LM (Meng et al., 2021) is superior to ELECTRA (Clark et al., 2020) and BERT (Devlin et al., 2019). We argue the reason is that COCO-LM is pre-trained on sequence contrastive learning with a dual-encoder architecture (Reimers and Gurevych, 2019), which is compatible for our CQR consistency tasks with dual-encoder for question context $q_{<\tau}$ and self-contained question $r_{\tau}$ as inputs.

<table>
<thead>
<tr>
<th>PLMs</th>
<th>Methods</th>
<th>SPARC$_{Dev}$ QM</th>
<th>SPARC$_{Dev}$ IM</th>
<th>CoSQL$_{Dev}$ QM</th>
<th>CoSQL$_{Dev}$ IM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>End-to-End</td>
<td>58.6</td>
<td>38.2</td>
<td>50.7</td>
<td>20.5</td>
</tr>
<tr>
<td></td>
<td>Two-Stage</td>
<td>60.1</td>
<td>39.3</td>
<td>51.1</td>
<td>22.2</td>
</tr>
<tr>
<td></td>
<td>CQR-SQL</td>
<td>62.5</td>
<td>42.4</td>
<td>53.5</td>
<td>24.6</td>
</tr>
<tr>
<td>ELECTRA</td>
<td>End-to-End</td>
<td>64.9</td>
<td>46.5</td>
<td>56.6</td>
<td>23.9</td>
</tr>
<tr>
<td></td>
<td>Two-Stage</td>
<td>65.8</td>
<td>46.7</td>
<td>56.8</td>
<td>24.6</td>
</tr>
<tr>
<td></td>
<td>CQR-SQL</td>
<td>67.8</td>
<td>48.1</td>
<td>58.2</td>
<td>29.4</td>
</tr>
<tr>
<td>COCO-LM</td>
<td>End-to-End</td>
<td>65.6</td>
<td>45.5</td>
<td>57.1</td>
<td>25.9</td>
</tr>
<tr>
<td></td>
<td>Two-Stage</td>
<td>66.0</td>
<td>46.5</td>
<td>57.8</td>
<td>26.3</td>
</tr>
<tr>
<td></td>
<td>CQR-SQL</td>
<td>68.0</td>
<td>48.8</td>
<td>58.5</td>
<td>31.1</td>
</tr>
</tbody>
</table>

Table 8: Results of End-to-End, Two-Stage and CQR-SQL methods with different PLMs.

---

https://pytorch.org/
B.5 More Cases

In this section, we show more cases of error propagation with Two-Stage pipeline method, and CQR-SQL against End-to-End baseline models.

<table>
<thead>
<tr>
<th>Cases of Error Propagation</th>
<th>❌ is the wrongly generated self-contained question.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: Show all models and horsepower of all car</td>
<td>❌</td>
</tr>
<tr>
<td>1: Now show just the ones with 4 cylinders.</td>
<td>❌</td>
</tr>
<tr>
<td>2: What is the model of the car with the lowest horsepower?</td>
<td>❌</td>
</tr>
<tr>
<td>3: What is the model of the car with the greatest horsepower?</td>
<td>❌</td>
</tr>
<tr>
<td>4: what is the model of that with the lowest horsepower?</td>
<td>❌</td>
</tr>
<tr>
<td>5: what is the greatest horsepower?</td>
<td>❌</td>
</tr>
<tr>
<td>6: Now show just the ones with 4 cylinders.</td>
<td>❌</td>
</tr>
<tr>
<td>7: how about the greatest horsepower?</td>
<td>❌</td>
</tr>
</tbody>
</table>

Two-Stage: SELECT model_list.Model FROM model_list JOIN cars_data ORDER BY cars_data.Horsepower DESC LIMIT 1

CQR-SQL: SELECT model_list.Model FROM car_names JOIN cars_data WHERE cars_data.Cylinders = "value" ORDER BY cars_data.Horsepower DESC LIMIT 1

0: List all the names of both Professionals and Dogs.

1: What about the dog names?

2: what are the names of dogs?

Two-Stage: SELECT Professionals.first_name FROM Professionals INTERSECT SELECT Owners.last_name FROM Owners

CQR-SQL: SELECT Dogs.name FROM Dogs

0: Show the name of dogs whose owners are from the city Lake Ti.

1: Add the owner's first names also.

2: What about the city of Virginita?

3: show the name of dogs whose owner is from the state of "virginia".

4: show the name of dogs whose owner is from the state of "virginia" and the owner's first names.

Two-Stage: SELECT Dogs.name, Owners.first_name FROM Dogs JOIN Owners WHERE Owners.state="value"

CQR-SQL: SELECT Dogs.name, Owners.first_name FROM Dogs JOIN Owners WHERE Owners.state="value"

Cases of CQR-SQL against End-to-end approach

0: Which countries have republics as their form of government?

1: Which language is spoken by only one of those countries?

Two-Stage: SELECT countrylanguage.Language FROM countrylanguage JOIN country GROUP BY countrylanguage.Language HAVING COUNT(*) = "value" ❌

CQR-SQL: SELECT countrylanguage.Language FROM countrylanguage JOIN country GROUP BY countrylanguage.Language HAVING COUNT(*) = "value" ❌

0: Find all employees who are under age 30.

1: Show the cities from which more than one employee originated.

Two-Stage: SELECT employee.City FROM employee GROUP BY employee.City HAVING COUNT(*) > "value" ❌

CQR-SQL: SELECT employee.City FROM employee WHERE employee.Age = "value" GROUP BY employee.City HAVING COUNT(*) > "value" ❌

Figure 7: Cases on SPARC dataset. Upper block shows the cases of error propagation with incorrectly generated self-contained questions $\tilde{r}_r$ for Two-Stage pipeline methods (as in Figure 2(c) or [7] in Table 3). Cases in the lower block show that End-to-End method (as in Figure 2(a) or [4] in Table 4) fails to resolve the conversational dependency.