# Faster and Better Grammar-based Text-to-SQL Parsing via Clause-level Parallel Decoding and Alignment Loss

Anonymous ACL submission

## Abstract

Grammar-based parsers have achieved high performance in the cross-domain text-to-SQL parsing task, but suffer from low decoding efficiency due to the much larger number of actions for grammar selection than that of tokens in SQL queries. Meanwhile, how to better align SQL clauses and question segments has been a key challenge for parsing performance. Therefore, this paper proposes clause-level parallel decoding and alignment loss to enhance two high-performance grammar-based parsers, i.e., RATSQL and LGESQL. Experimental results of two parsers show that our method obtains consistent improvements both in accuracy and decoding speed.

## 1 Introduction

Text-to-SQL parsing aims to automatically transform natural language (NL) questions into SQL queries based on the given databases (DBs) (Tang and Mooney, 2001), as depicted at the top of Figure 1. Recently, several high-quality cross-domain text-to-SQL datasets have been released, strongly boosting the research interest and progress in this task (Zhong et al., 2017; Yu et al., 2018b; Wang et al., 2020b). Most early works generate SQL queries in a token-level seq2seq manner (Zhong et al., 2017; Dong and Lapata, 2018), or by filling DB elements into SQL slots (Xu et al., 2017; Yu et al., 2018a), both of which are known as token-based parsers. In contrast, a grammar-based parser incorporates SQL grammar into the decoder to guarantee the grammaticality of output SQL queries (Yin and Neubig, 2018), including RATSQL (Wang et al., 2020a) and LGESQL (Cao et al., 2021), both of which have achieved state-of-the-art (SOTA) performance on complex datasets. They share the same decoder with different grammars, and LGESQL further uses a new graph encoder to enhance presentations of the question words and database schema items.

Concretely, a grammar-based parser builds a tree complying with SQL grammar via a sequence of actions, as shown on the left of Figure 1, where the tree's leaf nodes form the final SQL query. In spite of its high performance, the number of actions is usually much larger than the number of tokens in the SQL query, due to the generation of non-leaf nodes. This makes the decoding process extremely inefficient. To alleviate the inefficiency issue, DuoRAT (Scholak et al., 2021) uses a transformer-based decoder to replace the parent-feeding LSTM decoder in RATSQL (Wang et al., 2020a), which can improve the training efficiency given gold-standard SQL queries. Unfortunately, their method does not influence the testing speed, which is very important in real applications.

As discussed by many previous works, one characteristic of the text-to-SQL task is that an SQL clause usually depends on a local segment of the input question (Zeng et al., 2020; Yin et al., 2021; Wu et al., 2021). Recent works try to exploit alignments between SQL clauses and question segments for better handling some specific SQL structures. Zeng et al. (2020) propose a recursive parsing framework that can elegantly generate complicated nested SQL queries. The basic idea is explicitly encouraging the decoder to focus on different question segments when generating different nested layers. Based on a token-based parser, Yin et al. (2021) incorporate an extra attention loss to capture such alignments, which is proved to be helpful for dealing with compositional SQL queries.

To handle the above two issues, we propose to enhance grammar-based parsers via clause-level parallel decoding and alignment loss. First, we propose to generate SQL clauses in parallel, that is, clauses are generated independently of each other and simultaneously. Second, we propose a clause-level alignment training loss to encourage the model to focus on only related question segment when generating a clause. We implement these two strategies based on two SOTA grammar-based parsers, RATSQL and LGESQL. Experi-

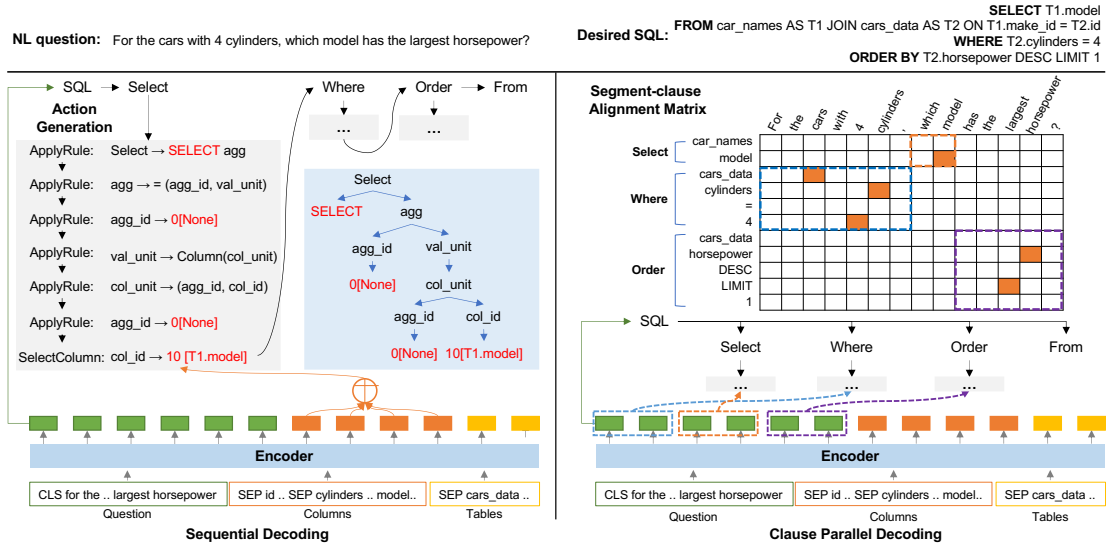


Figure 1: An overview of our approach. The left side shows the generation process of sequential decoding in RATSQL grammar-based decoder, and the right side gives our proposed parallel decoding, where all clauses are generated independently. Meanwhile, according to alignments between SQL clauses and question segments, as shown by the segment-clause alignment matrix, a clause-level alignment loss is incorporated during training.

mental results on Spider show that our methods obtain consistent improvements both in accuracy and testing speed. We will release our code at https://github.

## 2 Our Proposed Model

### 2.1 Grammar-based Text-to-SQL Parsing

As shown on the left side of Figure 1, the decoder generates SQL queries via generating actions to select grammar rules in the depth-first search order. Specifically, there are three types of actions, i.e., ApplyRule, SelectColumn and SelectTable. ApplyRule($r$) applies an action $r$ to expand the focus node, and is used to gradually create a skeleton tree without concrete DB elements. SelectColumn($c$) and SelectTable($t$) are used to fill a skeleton tree with concrete values by selecting a column name $c$ or a table name $t$, respectively.

We take an example in Figure 1 to illustrate the process of grammar-based decoder. Suppose that the decoder is at the "agg" node (current node, denoted as $n_i$) under the "Select" node (father node, denoted as $f_i$), and the next action is "ApplyRule(agg → agg_id val_unit)". The LSTM state of the decoder is updated as follows.

$$\begin{aligned} \mathbf{c}_{t+1}, \mathbf{h}_{t+1} &= \texttt{LSTM}(\mathbf{c}_t, \mathbf{h}_t, \mathbf{i}_t) \\ \mathbf{i}_t &= [\mathbf{e}_{a_t}; \mathbf{e}_{n_i}; \mathbf{e}_{\texttt{type}(n_i)}; \mathbf{z}_t; \mathbf{h}_{\texttt{tm}(f_i)}] \end{aligned} \quad (1)$$

where $\mathbf{c}_t$ and $\mathbf{h}_t$ are the cell state and the output vector at step $t$; $\mathbf{e}_n$ represents the embedding vector of the input $n$; $a_t$ denotes the previous action; $\texttt{type}(.)$ returns the type of a node[1]; $\mathbf{z}_t$ is the contextual representation vector after attending to the encoder outputs; $\texttt{tm}(f_i)$ denotes the timestamp when $f_i$ has been just generated.

### 2.2 Clause-level Parallel Decoding

During decoding, the grammar-based parser actually generates a SQL query by sequentially creating clauses (seeing Table 1) in a predefined order, as shown on the left side of Figure 1. For instance, after completing the SELECT clause, the parser tries to expand the WHERE clause. If "WHERE → None" is selected by the decoder, it means that the final SQL query does not include a WHERE clause and the decoder will move on to generate the GROUP clause, and so on. In fact, the generation of different clauses is quite loosely connected. This motivates us that we may generate all SQL clauses independently and in parallel via batch processing, which obviously can improve decoding efficiency.

Specifically, major differences between parallel and sequential decoding lie in the initial LSTM state of each clause, reflected in $\mathbf{c}_0$, $\mathbf{h}_0$, and the previous action $a_0$ in Equation 1. In sequential decoding, the initial status for a subsequent clause is inherited from and thus depends on the previous clause. In contrast, in parallel decoding, each

[1]The parser assigns a type for each node according to its role in SQL, such as "agg" for aggregations.

| Clauses | Production Rules |
|---|---|
| SELECT | SELECT agg \| SELECT agg, agg \| ... |
| WHERE | WHERE \| None |
| GROUP | GROUP_BY \| GROUP_BY & HAVING \| None |
| ORDER | ORDER_BY \| ORDER_BY & LIMIT \| None |
| IEU | INTERSECT \| EXCEPT \| UNION \| None |
| FROM | FROM Table1 \| FROM Table1, Table2 \| ... |

Table 1: Six common types of clauses used for parallel decoding in our work.

clause has the same initial status, which we believe is more reasonable considering the loose dependency between adjacent clauses.

### 2.3 Clause-level Alignment Loss

Figure 1 shows the alignment between question segments and SQL clauses. We use this alignment to improve clause generation by introducing a clause-level alignment training loss to encourage the model to focus on the aligned question segment in the clause generation.

**Clause-level Alignment Acquisition.** Given a question/SQL pair, for each DB element and condition value in the SQL query, we search for some tokens from NL question to align them, so as to get a token-level alignment matrix. In this process, we use the string-matching method which is commonly used for token-level schema linking in recent works (Guo et al., 2019; Wu et al., 2021). As shown in the alignment matrix in Figure 1, token-level alignments are marked in orange box.

Then we use a simple heuristic algorithm to extract a question segment for each SQL clause from existing token-level alignment results. For each clause, we take the shortest question segment that contains all DB elements and values in the clause as its aligned segment. As shown in the alignment matrix of Figure 1, the question segment for a clause is marked by a dashed bounding box. Please note that the question segments for different clauses may have overlaps. Finally, there are about 23% question/SQL pairs missing segment-clause alignments. For these pairs, we align each clause to the whole question. We believe that higher-quality alignment may lead to higher gains, which we leave as future work.

**Clause-level Alignment Loss.** After aligning SQL clauses with NL question segments, we design an extra training loss to inject such clause-level alignment into the parsing model. Intuitively, the model can be benefited by paying more attention to related aligned segments during clause generation.

In our grammar-based parser, a clause is generated by a sequence of actions. For instance, the SELECT clause in Figure 1, i.e., "select T1.model", which is aligned to "which model", is generated by six ApplyRule actions and one SelectColumn action. For each ApplyRule($r$) action, we define a prior token-wise alignment probability towards its corresponding segment[2]. Concretely, each token in the segment obtains an averaged probability, whereas tokens outside the segment receive zero.

$$P_{\texttt{align}}(x_i|r_j) = \begin{cases} 0 & x_i \notin S \\ 1/\texttt{len}(S) & x_i \in S \end{cases}$$

where $r_j$ is the rule in the ApplyRule action, and $x_i$ is the $i$-th token in the sentence, and $\texttt{len}(S)$ is the number of tokens in the aligned segment $S$.

Then, we define an attention probability from the current decoder state to each question token as

$$P_{\texttt{att}}(x_i|r_j) = \texttt{softmax}(..., \mathbf{h_t} W_m \mathbf{m_i}, ...)$$

where $t$ is the timestamp when executing ApplyRule($r_j$); $\mathbf{h_t}$ is the time $t$'s hidden state of LSTM decoder; $\mathbf{m_i}$ is the output vector for $x_i$ in encoder outputs; $W_m$ is a learned matrix.

Finally, we define the alignment loss as the squared distance between the aligned (prior) and attention (modeling) probabilities.

$$L = \sum_j \sum_i (P_{\texttt{align}}(x_i|r_j) - P_{\texttt{att}}(x_i|r_j))^2$$

In this way, we hope the model learn to attend to certain related question tokens for the sake of better rule selection.

## 3 Experiments

**Dataset.** We conduct experiments on Spider[3], a complex and cross-domain text-to-SQL dataset. We follow the original data splitting and use the exact matching (EM) accuracy as the evaluation metric. In our experiments, we use the corrected development set released on June 7, 2020.

**Implementation.** We implement our proposed strategies on RATSQL and LGESQL. The final loss for the training model is the summation of the

[2]We don't use alignment loss for other two actions, since they tend to be closely related with one or two tokens in NL question. Forcing such action to align with too many tokens in a segment degrades the performance, which has been proved by our early-stage preliminary experiments.

[3]https://yale-lily.github.io/spider

| Models | EM Accuracy | Parsing Speed (query/second) |
|---|---|---|
| DuoRAT + BERT (Duo2021) | 69.9 | - |
| RATSQL (Wang2020) | | |
| + BERT | 69.7 | - |
| + GRAPPA | 73.4 | - |
| LGESQL(Cao2021) | | |
| + BERT | 73.5 | - |
| + GRAPPA | 74.1 | - |
| + ELECTRA | 75.1 | - |
| **RATSQL** | | |
| Orig. + BERT (rerun) | $71.1_{\pm 0.4}$ | 7.48 |
| Ours + BERT | $72.5_{\pm 0.1}$ | 9.14 (+18.4%) |
| w/o Align | $71.7_{\pm 0.2}$ | 9.21 (+18.9%) |
| w/o Parallel | $72.4_{\pm 0.1}$ | - |
| Ours + GRAPPA | $74.2_{\pm 0.4}$ | - |
| **LGESQL** | | |
| Orig. + ELECTRA (rerun) | $75.1_{\pm 0.7}$ | 11.69 |
| Ours + ELECTRA | $75.7_{\pm 0.6}$ | 15.81(+35.2%) |
| w/o Align | $75.3_{\pm 0.6}$ | 15.84(+35.5%) |
| w/o Parallel | $75.6_{\pm 0.4}$ | - |

Table 2: EM accuracy and testing speed on Spider dev set. For our models, we report mean and variance over three runs.

| RATSQL | Easy | Medium | Hard | Extra Hard |
|---|---|---|---|---|
| Orig. + BERT | 87.9 | 72.9 | 63.2 | 49.4 |
| Ours + BERT | 88.7 | 74.9 | 64.9 | 50.0 |

Table 3: EM accuracy on different hardness levels.

original loss and our proposed alignment loss. We set beam size as 1 to evaluate testing speed, and use default values for other parameters.

**Main results.** Table 2 shows the main results. In the first major row, we select several BERT (Devlin et al., 2019) enhanced grammar-based parsers. We report our results in the second and third major rows[4]. Besides using BERT, we also give the results with GRAPPA(Yu et al., 2020), a task-specified pre-trained model. For LGESQL, we give results with ELECTRA(Clark et al., 2020), which achieves SOTA performance. In order to avoid the effect of performance vibrations, we run each model for 3 times with different random initialization seeds, then report the averaged EM accuracy and the variance.

*Parallel decoding*. The parallel decoding achieves an average accuracy improvement of 0.6% and 0.2% for RATSQL and LGESQL, which proves that there is no strong generation dependency between SQL clauses. Meanwhile, the parallel decoding improves parsing speed by 18.9% and 35.5% for two parsers, as shown in the last column of Table 2. For LGESQL, the parallel decoding achieves a larger improvement in parsing speed, as

[4]We have submitted our models for obtaining results on test set. However, due to some environmental problems and limited GPU resources, the results have not been returned.

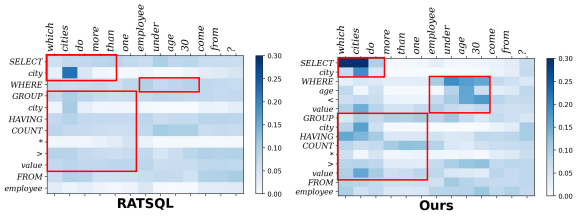


Figure 2: Visualization of RATSQL attention scores. Red rectangles highlight alignment blocks that obtain high scores in our model but low scores in baseline.

its grammar is simpler and the action sequence for each clause is much shorter. The parallel decoding is more effective when there is little difference in action sequence length for all clauses.

*Effect of alignment loss*. The clause-level alignment loss can improve two models by 1.3% and 0.5%, although its incorporation slightly decreases the testing speed. Integrating the two strategies bring slight improvements compared with alignment loss. We suspect that the clause-level alignment loss has weakened the generation dependency between clauses by encouraging the model to focus on the aligned segments when generating clauses.

**Hardness analysis**. As shown in Table 3, our model outperforms original RATSQL on all hardness levels. Yu et al. (2018b) define the SQL hardness based on the number of clauses and the number of components in a clause. Our method obtains larger gains on harder examples[5], i.e, "Medium" and "Hard", in which 60% of the SQL queries contain no less than three clauses.

**Case study**. In order to verify the impact of clause-level alignments in the attention mechanism, we plot attention weights of original RATSQL and our RATSQL in Figure 2. In our model, each clause has a higher attention weight with tokens in the corresponding aligned segment. Inversely, the base model doesn't have focus attention scores for some clauses, such as WHERE and GROUP, and it fails to generate the WHERE clause.

## 4 Conclusion

We propose clause-level parallel decoding and alignment loss to enhance grammar-based text-to-SQL parsing models. Experimental results show that our approach improves consistently both their efficiency and accuracy.

[5]The SQL query in "Extra Hard" level usually requires common knowledge and involves logical reasoning. The base model and our method are still far from resolving these.

## References


Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. Lgesql: Line graph enhanced text-to-sql model with mixed local and non-local relations. *arXiv preprint arXiv:2106.01093*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742.

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 *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535.

Torsten Scholak, Raymond Li, Dzmitry Bahdanau, Harm de Vries, and Christopher Pal. 2021. Duorat: Towards simpler text-to-sql models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1313–1321.

Lappoon R Tang and Raymond J Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *European Conference on Machine Learning*, pages 466–477. Springer.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020a. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.

Lijie Wang, Ao Zhang, Kun Wu, Ke Sun, Zhenghua Li, Hua Wu, Min Zhang, and Haifeng Wang. 2020b. DuSQL: A large-scale and pragmatic Chinese text-to-SQL dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6923–6935, Online. Association for Computational Linguistics.

Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data augmentation with hierarchical SQL-to-question generation for cross-domain text-to-SQL parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8974–8983, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Pengcheng Yin, Hao Fang, Graham Neubig, Adam Pauls, Emmanouil Antonios Platanios, Yu Su, Sam Thomson, and Jacob Andreas. 2021. Compositional generalization for neural semantic parsing via span-level supervised attention. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2810–2823.

Pengcheng Yin and Graham Neubig. 2018. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 7–12.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.

Yu Zeng, Yan Gao, Jiaqi Guo, Bei Chen, Qian Liu, Jian-Guang Lou, Fei Teng, and Dongmei Zhang. 2020. Recparser: A recursive semantic parsing framework for text-to-sql task. In *Twenty-ninth International Joint Conference on Artificial Intelligence*, pages 3644–3650.

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