Testing Cross-Database Semantic Parsers Using Canonical Utterances

Heather Lent[ç]* **Semih Yavuz**[ʃ] **Tao Yu**[ﬁ]
Tong Niu[ʃ] **Yingbo Zhou**[ʃ] **Dragomir Radev**[Y] **Xi Victoria Lin**[ɸ]
[ç] University of Copenhagen, [ʃ] Salesforce Research
[ﬁ] Hong Kong University, [Y] Yale University, [ɸ] Facebook AI Research
`hcl@di.ku.dk, tyu@cs.hku.hk`
`{syavuz, tniu, yingbo.zhou}@salesforce.com`
`dragomir.radev@yale.edu, victorialin@fb.com`

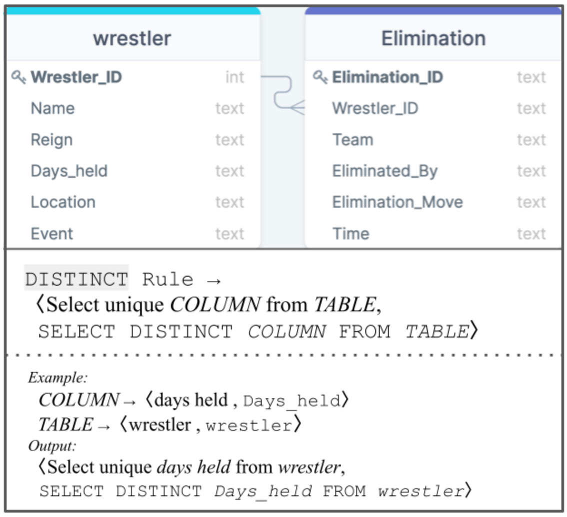
Abstract

The benchmark performance of cross-database semantic parsing has climbed steadily in recent years, catalyzed by the wide adoption of pre-trained language models. Yet existing work have shown that state-of-the-art cross-database semantic parsers struggle to generalize to novel user utterances, databases and query structures. To obtain transparent details on the strengths and limitation of these models, we propose a diagnostic testing approach based on controlled synthesis of canonical natural language and SQL pairs. Inspired by the CheckList (Ribeiro et al., 2020), we characterize a set of essential capabilities for cross-database semantic parsing models, and detailed the method for synthesizing the corresponding test data. We evaluated a variety of high performing models using the proposed approach, and identified several non-obvious weaknesses across models (e.g. unable to correctly select many columns). Our dataset and code are released as a test suite at `http://github.com/hclent/BehaviorCheckingSemPar`.

1 Introduction

Cross-database semantic parsing, the task of mapping natural language utterances to SQL queries for any database, has attracted increasing attention since the introduction of benchmarks like WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018). The advent of pre-trained language models (Peters et al., 2018; Devlin et al., 2019; Liu et al., 2019; Lewis et al., 2020) has further accelerated the progress in this area (Lin et al., 2020; Yu et al., 2020; Shi et al., 2020; Wang et al., 2020; Choi et al., 2020).

Despite impressive gains on standard benchmarks, studies on cross-database semantic parsing models show that they still suffer from out-of-distribution (OOD) generalization when presented with novel user utterances (Suhr et al., 2020; Radhakrishnan et al., 2020; Shaw et al., 2021), databases (Suhr et al., 2020) and SQL query structures (Finegan-Dollak et al., 2018; Suhr et al., 2020; Shaw et al., 2021). As baseline performance climbs ever upward, at what point can we confidently deploy our models to end users, and how will we know we have reached this point?



`DISTINCT` Rule →
⟨Select unique *COLUMN* from *TABLE*,
`SELECT DISTINCT` *`COLUMN`* `FROM` *`TABLE`*⟩

Example:
COLUMN → ⟨days held , `Days_held`⟩
TABLE → ⟨wrestler , `wrestler`⟩
Output:
⟨Select unique *days held* from *wrestler*,
`SELECT DISTINCT` *`Days_held`* `FROM` *`wrestler`*⟩

Figure 1: The database (top) is applied to our SCFG production rule (middle) to produce a new example for the `DISTINCT` category (bottom). See Appendix B for production rules of other categories.

Inspired by Ribeiro et al. (2020), which has shown the effectiveness of simple, systematic, and heuristic behavior checking strategies for evaluating the robustness of NLP models, we propose a controllable, non-adversarial unit testing approach to shed more light on the capabilities of cross-database semantic parsers. We implement a synchronous context-free grammar (SCFG) to generate natural language questions based on SQL queries (Figure 1). This grammar features production rules that evaluate important categories of SQL element types such as clauses (e.g. `SELECT` and `WHERE`), as well as commonly used operators including aggre-

*Work done during an internship at Salesforce Research.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 73–83
July 5–10, 2020. ©2020 Association for Computational Linguistics
https://doi.org/10.26615/978-954-452-056-4_008

gators (MAX), conditionals (BETWEEN), and logical operators (OR). We handcraft the rules for these categories to ensure that the generated question-query pairs are simple, natural, unambiguous, and with minimal cross-category overlap.

We apply our evaluation framework to four state-of-the-art text-to-SQL models, namely BRIDGE (Lin et al., 2020), RATSQL-RoBERTa and RATSQL-GraPPa (Yu et al., 2020), and RATSQL-GAP (Shi et al., 2020), and observe that these models struggle to extend their success on the Spider dev set consistently to our evaluation data, with the exception of a few categories. Further analysis of the fine grained categories shows that they also fail on many rudimentary test cases (e.g., selecting multiple columns and properly producing conjunctions). While existing studies show that the models tend to fail on challenging cases that involve novel user expression (Suhr et al., 2020) and SQL structures (Suhr et al., 2020; Shaw et al., 2021), our diagnosis exposes more robustness issues in their surface form understanding (even with seemingly simple inputs), and highlights the importance of addressing such issues in the modeling foundation (Bommasani et al., 2021). Our dataset and code are released as an extensible test suite.

2 Related Work

Paraphrasing A number of augmentation methods have been made to create paraphrases of the input query, with methods such as synonym replacement (Kwiatkowski et al., 2013), use of a paraphrase model (Berant and Liang, 2014), and backwards utterance generation (Zhong et al., 2020). While these approaches ensure the creation of additional examples with more variation on the natural language side, they can be vulnerable to error, when a wrong synonym or paraphrase is chosen by a model. Although such errors may amount to just noise when used as additional training data in conjunction with a benchmark dataset, they make evaluation on such generated sets impossible, unless examples with errors are manually removed from the dataset.

Canonical Utterances Wang et al. (2015) demonstrated that it is possible to lessen the reliance on humans for creating a dataset by first generating logical forms and canonical utterances, and then use crowdsourcing to create more natural-sounding paraphrases of the questions. They note that this method is particularly effective when you seek to quickly create data for creating a *domain specific* parser. Iyer et al. (2017) also demonstrated that crowdsourced annotations from such approaches, as in turn user feedback in an online setting, can be used improve parses and detect incorrect queries. Although originally designed in the context of transfer-based machine translation to generate translation pairs (Chiang, 2005), SCFG's have also been adapted in previous semantic parsing work (Wong and Mooney, 2006, 2007) for generating new sentence-parse pairs. More recent utilization's of SCFG's for semantic parsing induce the grammar and use the resulting data for additional training and pre-training (Jia and Liang, 2016; Yu et al., 2020).

Robustness Testing Finally, Ribeiro et al. (2020) has demonstrated the efficacy of handcrafting templates for generating data points to "unit test" the models. We design synchronous context-free grammar (SCFG) production rules to generate test data for specific cross-database semantic parsing capabilities. Other NLP evaluation frameworks that look beyond accuracy and target a more general set of NLP tasks have also been proposed (Goel et al., 2021; Liu et al., 2021; Kiela et al., 2021).

3 Generating Canonical Natural Language Utterances Using SCFG

Motivation There are in general two ways to perform behavior testing on a model: one with automatically generated data, the other with manually curated data. In this work we focus on the former because it not only scales with almost no additional cost, but also serves as a pre-filtering mechanism before we test it further with human-in-the-loop. The input to text-to-SQL models is a *natural* question. However, generating natural language has two challenges: (i) it is difficult to automatically produce novel human-like utterances with high-fidelity; (ii) natural language is inherently ambiguous, while input to text-to-SQL models is required to be accurate enough to have a one-to-one mapping between the natural question and the SQL query. Motivated by the above requirements, we propose using the inherently non-ambiguous **S**ynchronous **c**ontext-**f**ree **g**rammar (SCFG) for generating canonical natural language utterances in English[1].

[1] This method is also extendable to other languages.

Details of SCFG SCFG is a type of formal grammar which produce pairs of utterances that share a meaning with each other. There are two key components of a context-free grammar: *symbols* and *production rules* that connect them. In our case, the symbols correspond to the SQL elements, which are presented in the first column of Table 1.[2] The production rules are mappings between SQL elements and natural language words. In Figure 1 we provide such an example where SCFG maps the SQL element `DISTINCT` to the word "*unique*", hence converting the SQL query "SELECT DISTINCT Column FROM Table" to the natural language question "Select unique Column from Table". The mappings between symbols and query words are intentionally designed to mimic the language in the Spider dataset (Yu et al., 2018), which ensures that the generated examples remain close to the training distribution.[3]

Intuitively, questions produced by the SCFG lie somewhere in-between natural language and SQL: they are not as natural as real human questions, but are much more human-like than the SQL queries. Accommodating such a trade-off ensures that the generated queries are both natural and accurate. More examples of SCFG rules can be found in Appendix C.

Generation of evaluation data To thoroughly evaluate each SQL element, we create as many valid question-query pairs as possible for each database in Spider, so that there is adequate representation for infrequent categories. Note that many databases have tables that only correspond to a subset of elements.[4] Consequently the number of collected examples in Table 1 (second column) are not evenly distributed.[5]

When generating examples for a given SQL element, the example operates over only one table, and we only introduce the minimum amount of other elements to make the generation grammatical and uncompounded. For example, the operator `BETWEEN` necessitates `SELECT` and `WHERE` clauses to generate a coherent query, but any additional operators, even if they can make the query more compositional, are excluded, as our goal is to unit test each SQL element individually. In turn, our generated data are also intended to be as easy as possible for models to succeed on.

			Exact Set Match Acc.			
	Target SQL Element	**#**	BRIDGE†	RATSQL+		
				RoBERTa	GraPPa	GAP
	Spider Dev	1034	68.2	69.6	73.4	71.8
Basic Clauses	`SELECT`	1700	53.6	46.5	62.6	73.5
	`DISTINCT`	850	86.4	86.6	94.5	88.3
	`WHERE`	1003	73.2	70.3	84.4	82.1
	`ORDER BY`	1946	51.0	54.7	71.4	76.5
	`GROUP BY`	653	35.5	51.3	45.9	5.7
	`HAVING`	604	0.1	0.0	0.0	0.0
	Cat. Avg.		53.4	53.7	**65.7**	64.4
Aggregate Ops	`MIN`	794	74.5	59.1	93.7	83.2
	`MAX`	794	75.3	17.5	85.9	47.4
	`SUM`	794	66.0	71.1	52.2	52.1
	`COUNT`	850	34.4	56.3	70.3	66.8
	`AVG`	794	56.7	58.1	81.8	79.7
	Cat. Avg.		61.0	52.5	**76.7**	65.9
Condition Ops	$\leqslant, <, >, \geqslant$	440	55.2	37.9	61.3	88.6
	$!=$	397	27.2	68.3	62.4	92.4
	`BETWEEN`	256	65.9	26.7	34.9	51.0
	Cat. Avg.		49.4	44.3	52.9	**77.3**
Logic Ops	`AND`	401	3.2	4.5	7.2	16.2
	`OR`	401	5.1	5.0	8.2	17.1
	`AND & OR`	369	4.1	4.3	8.6	18.1
	Cat. Avg.		4.1	4.6	8.0	**17.1**
	Overall Avg.		45.0	42.9	55.3	**55.6**

Table 1: Results on the models per our SCFG categories. # shows the number of test examples present. Cat. Avg. reflects the category average weighted by the number of examples per each target SQL element. †BRIDGE results are averaged across three checkpoints with different random initializations, while the RAT-SQL results are based on the best checkpoints according to the dev set evaluation.

Human verification of evaluation data To verify that our generated examples are indeed human-like and accurate, we recruited volunteers[6] who are proficient in SQL to label a subset of 40 randomly chosen question-query pairs, and rate each pair on its "readability" and "semantic equality". The question-query pairs are chosen such that all cate-

[2] We collected the SQL elements from `https://www.w3schools.com/sql/` and `https://www.techonthenet.com/sqlite/`.

[3] Competent performance across categories in Table 1 demonstrate our data overlap with the training distribution.

[4] For example, a table with only text-type columns can not be used to generate pairs with mathematical concepts *minimum* or *less than*.

[5] To have a uniform distribution, one may perform subsampling (which wastes valuable data), or design a model to automatically generate new tables – we leave the latter as future work.

[6] Our annotation task posed no risk or harm to annotators, and required 30 minutes of the volunteers' time.

	Target SQL Element and Example	Model Predictions with Highlighted Errors
SELECT		***BRIDGE:***`SELECT student.ID, student.name, student.dept_name, student.tot_cred FROM student`
	NL: Select name, id, department name, total credits from student	***RS+RoB:***`SELECT student.name, student.ID, student.dept_name, Sum(student.tot_cred) FROM student GROUP BY student.ID`
	SQL:`SELECT name, ID, dept_name, tot_cred FROM student`	***RS+GraPPa:***`SELECT student.name, student.ID, student.dept_name, Sum(student.tot_cred) FROM student`
		RS+GAP:`SELECT student.name, student.ID, student.dept_name, Sum(student.tot_cred) FROM student`

Table 2: Model predictions on a randomly chosen SELECT example. See Appendix B for additional qualitative examples of model predictions on different categories.

Columns	#	Exact Set Match Acc.			
		BRIDGE	RATSQL+		
			RoBERTa	GraPPa	GAP
1	852	69.1	52.3	70.8	**85.4**
2	253	60.9	68.8	81.0	**88.9**
3	191	68.3	63.4	**85.9**	**85.9**
4	154	21.0	32.5	61.0	**81.8**
5	122	0.0	0.0	0.0	0.0
6	69	0.0	0.0	0.0	0.0

Table 3: Performance of models on SELECT clauses by number of columns being selected.

gories are represented at least twice. Each question-query pair was annotated by three annotators and we take their majority vote. An example given to annotators can be found in Appendix B.

For readability, 77.5% of generated questions were labeled by annotators to be "easily understandable"; 17.5% were labeled "understandable with some effort" and 5% were labeled "not understandable". We obtain this statistics by taking the majority vote of the three annotations for each question and counting a tie as "not understandable". In the same manner, annotators also identified 97.5% of questions were "semantically equivalent" to their SQL counterpart and 2.5% were "not equivalent".

We computed Fleiss' Kappa to measure inter-annotator agreement for both readability and equivalency. The results were 0.19 and 0.04, respectively, which are generally considered insufficient to claim there is strong agreement. However, we find the low scores a result of the limitation of Fleiss' Kappa, which is more reliable when each example is annotated by more annotators (we have only 3). Reviewing the annotations for readability reveals that there were 14 examples without perfect agreement for readability. For equivalency, there were only 4 of them.

4 Experiments

4.1 Experiment Setup

Models We evaluate four leading models on the Spider challenge (Yu et al., 2018) on our generated question-query pairs: BRIDGE (Lin et al., 2020), RATSQL-RoBERTa and RATSQL-GraPPa (Yu et al., 2020) and RATSQL-GAP (Shi et al., 2020). With the exception of BRIDGE, the other models were developed upon the original RATSQL model (Wang et al., 2020), which was notable for introducing a *relation-aware self-attention mechanism* for schema linking. Yu et al. (2020) extended the RATSQL framework by adding pre-training into their setup, and Shi et al. (2020) also incorporates supplementary pre-training triplet data generated by another model. The BRIDGE model is fundamentally different from the others, as it consists of a sequentially-driven architecture, rather than operating over graphs. For schema-linking, BRIDGE uses a custom encoder powered by BERT (Devlin et al., 2019) with attention over the sequences.

Evaluation Methodology Our experiments consist of evaluating each model on the generated set of question-query pairs with the canonical language questions as inputs. We evaluate Exact Set Match Accuracy for subsets of the data pertaining to each target SQL element, and then calculate the average score for each SQL token category weighted by number of examples.

4.2 Results

Main Results Table 1 highlights several interesting observations.[7] Most models only perform on par with their baseline (or better) on a few target SQL elements (e.g. DISTINCT, WHERE). More

[7] The metrics in Table 1 are diagnostic instead of explanatory. There can be multiple factors affecting the model performance on an evaluation point and our tests cannot isolate them.

often they perform below the baseline on most elements, with a few extreme outliers for total or near total failure (e.g. `HAVING`, `AND`).

Controlled Evaluation All models perform below their own baseline accuracies for simple examples that test the `SELECT` clause. We present an example of such model predictions in Table 2. One contributing factor to these low scores is the number of columns being selected. Table 3 shows that SQL models are only able to successfully produce queries with a limited number of columns, although basic column selection should not be such a difficult task for these models. While it is not surprising that models show difficulty generalizing to unseen length or structures (Lake and Baroni, 2017), this finding is concerning because there are many practical use cases where users will need to select more than four columns.[8]

5 Conclusion

We propose a simple and controllable approach for synthesizing text-to-SQL pairs for unit testing model performance on various semantic categories. Our controlled test suites allow for more extensive and fine-grained evaluation of state-of-the-art text-to-SQL models, which reveal a general lack of robustness in generalizing beyond the benchmark examples across several categories such as `SELECT` and `WHERE`. More importantly, our study highlights the importance of developing evaluation strategies beyond fixed test and dev set accuracy for understanding real progress made by the state-of-the-art text-to-SQL models and the remaining key challenges.

6 Acknowledgments

We would like to thank our reviewers for their helpful feedback. Heather Lent received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 801199.

References

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland. Association for Computational Linguistics.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. 2021. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 263–270, Ann Arbor, Michigan. Association for Computational Linguistics.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Karan Goel, Nazneen Fatema Rajani, Jesse Vig, Samson Tan, Jason Wu, Stephan Zheng, Caiming Xiong, Mohit Bansal, and Christopher Ré. 2021. Robustness gym: Unifying the NLP evaluation landscape. *CoRR*, abs/2101.04840.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.

[8] For example, the large tables in Spider's *soccer_1* database

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, Zhiyi Ma, Tristan Thrush, Sebastian Riedel, Zeerak Waseem, Pontus Stenetorp, Robin Jia, Mohit Bansal, Christopher Potts, and Adina Williams. 2021. Dynabench: Rethinking benchmarking in NLP. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 4110–4124. Association for Computational Linguistics.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556, Seattle, Washington, USA. Association for Computational Linguistics.

Brenden M. Lake and Marco Baroni. 2017. Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks. *CoRR*, abs/1711.00350.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online. Association for Computational Linguistics.

Pengfei Liu, Jinlan Fu, Yang Xiao, Weizhe Yuan, Shuaichen Chang, Junqi Dai, Yixin Liu, Zihuiwen Ye, and Graham Neubig. 2021. EXPLAINABOARD: an explainable leaderboard for NLP. *CoRR*, abs/2104.06387.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Karthik Radhakrishnan, Arvind Srikantan, and Xi Victoria Lin. 2020. Colloql: Robust cross-domain text-to-sql over search queries. *CoRR*, abs/2010.09927.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 922–938. Association for Computational Linguistics.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2020. Learning contextual representations for semantic parsing with generation-augmented pre-training.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online. Association for Computational Linguistics.

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

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Yuk Wah Wong and Raymond Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language*

Technology Conference of the NAACL, Main Conference, pages 439–446.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967.

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.

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

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

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

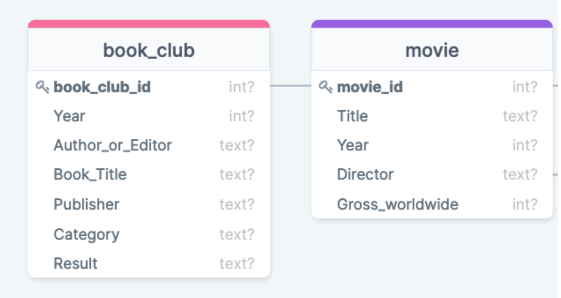
A Model Performance on Dev Examples Corresponding to Categories

Target SQL Element	#train	#dev	Exact Set Match Acc. BRIDGE	RATSQL+ RoBERTa	 GraPPa	 GAP
`SELECT`	213	32	82.3	90.6	96.9	81.2
`DISTINCT`	113	5	86.7	100	100	60.0
`WHERE`	343	61	77.2	83.6	83.6	100
`ORDER BY`	560	83	78.3	88.0	90.4	78.3
`GROUP BY`	16	8	83.3	50.0	50.0	75.0
`MIN`	2	4	16.7	0.0	50.0	0.0
`MAX`	10	5	0.0	0.0	0.0	0.0
`SUM`	25	2	100	100	100	100
`COUNT`	245	40	99.2	100	97.5	97.5
$<=, <, >, >=$	70	6	77.8	66.7	100	100
$! =$	14	52	83.3	85.7	85.7	100
`AND`	50	5	66.7	60.0	100	60.0
`OR`	54	10	100	88.0	100	78.3

Table 4: Performance of models on Spider Dev by our categories. SCFG elements that had zero corresponding examples are removed from the table. Here we include the number of examples in Spider training and Spider dev to demonstrate the underlying training and development distributions. Examples counted here are strictly relate to the chosen category. (i.e. examples with multiple SQL elements that do not pertain exactly to the categories are excluded from these counts).

B Example of Annotation Task

Example database schema given to annotators:



book_club	
book_club_id	int?
Year	int?
Author_or_Editor	text?
Book_Title	text?
Publisher	text?
Category	text?
Result	text?

movie	
movie_id	int?
Title	text?
Year	int?
Director	text?
Gross_worldwide	int?

Example question-query pair given to annotators: **/Question:** Select year from movie when movie id is greater than 1
\Query: `SELECT Year FROM movie WHERE movie_id > 1 ;`

Annotators are asked to choose one answer from the list below, to describe the readability and equivalency of the question-query pair, above:

1. **Readability**:
   - I can easily understand the question

- I have some problems understanding the question, but I can understand with some effort
- I do not understand the question after trying my best to interpret it

2. **Equivalency**:

   - The question and the SQL query match perfectly
   - The question and SQL query do not fully match, but the answer to the question can be inferred from the SQL query results
   - The SQL query does not return the answer to the question

C Example Model Predictions and SCFG Production Rules

See Table 5 for model predictions, and Figures 2 and 3 for example SCFG production rules.

	Target SQL Element and Example	Model Predictions with Highlighted Errors & Highlighted Missing Elements
GROUP_BY	*NL:* Select time slot id and minimum end minute from time slot grouped by time slot id *SQL:* SELECT time_slot_id, MIN(end_min) FROM time_slot GROUP BY time_slot_id	*BRIDGE:* SELECT time_slot.time_slot_id, MIN(time_slot.end_hr) FROM time_slot GROUP BY time_slot.time_slot_id *RS+RoB:* SELECT time_slot.time_slot_id, MIN(time_slot.end_min) FROM time_slot GROUP BY time_slot.time_slot_id *RS+GraPPa:* SELECT time_slot.time_slot_id, Min(time_slot.end_min) FROM time_slot GROUP BY time_slot.time_slot_id *RS+GAP:* SELECT time_slot.time_slot_id, Min(time_slot.end_min), Min(time_slot.end_min) FROM time_slot GROUP BY time_slot.time_slot_id
HAVING	*NL:* Select name from people grouped by name with maximum people id equal to 2 *SQL:* SELECT Name FROM people GROUP BY Name HAVING MAX(People_ID) = 2	*BRIDGE:* SELECT people.Name, MAX(people.Height) FROM people JOIN perpetrator ON people.People_ID = perpetrator.People_ID WHERE people.People_ID = 2 GROUP BY people.People_ID people.Name HAVING MAX(people.People_ID) =2) *RS+RoB:* SELECT people.Name FROM people GROUP BY people.Name HAVING Count(*) >= MAX(people.People_ID =) ’terminal’ *RS+GraPPa:* SELECT people.Name, Max(people.People_ID) FROM perpetrator JOIN people ON perpetrator.People_ID = people.People_ID GROUP BY people.Name HAVING Count(*) >= MAX(people.People_ID) = ’terminal’ *RS+GAP:* SELECT people.Name FROM people JOIN perpetrator WHERE GROUP BY Name HAVING MAX people.People_ID = ’terminal’
OR	*NL:* Select building from classroom when building equals Stabler or capacity equals 115 or room number equals 261 *SQL:* SELECT building FROM classroom WHERE building = ’Stabler’ OR capacity = 115 OR room_numer = 261	*BRIDGE:* SELECT classroom.building FROM classroom WHERE classroom.building = ’Stabler’ OR classroom.capacity = 115 UNION SELECT * FROM classroom WHERE classroom.building = ’Stabler’ OR classroom.room_numer = 261 *RS+RoB:* SELECT classroom.building FROM classroom WHERE classroom.building = ’terminal’ OR classroom.capacity = ’terminal’ OR classroom.room_number = 261 *RS+GraPPa:* SELECT classroom.building FROM classroom WHERE classroom.building = ’terminal’ OR classroom.capacity = ’terminal’ OR classroom.room_numer = 261 *RS+GAP:* SELECT classroom.building FROM classroom WHERE classroom.building = ’terminal’ OR classroom.capacity = ’terminal’ AND OR classroom.room_number = ’terminal’ AND classroom.room_number = ’terminal’

Table 5: Example predictions on selected target SQL elements from the BRIDGE, and RATSQL (RS) based models using RoBERTa (+RoB), GraPPa, and GAP.

SELECT

Rule → ⟨ Select Column from Table, SELECT Column FROM Table⟩

Example:
Table → ⟨ school performance , school_performance ⟩
Column → ⟨ class a , Class_A ⟩
Output Production:
NL: Select class a from school performance
SQL: SELECT Class_A FROM school_performance

Rule → ⟨ Select Columns from Table, SELECT Columns FROM Table⟩

Example:
Table → ⟨ people , people ⟩
Columns → ⟨ (height, name, weight, people id) , (Height, Name, Weight, People_ID) ⟩
Output Production:
NL: Select height, name, weight, people id from people
SQL: SELECT Height, Name, Weight, People_ID FROM people

ORDER_BY

Rule → ⟨ Select Column1 from Table sorted by Column2,
SELECT Column1 FROM Table ORDER BYColumn2⟩

Example:
Table → ⟨ circuits , circuits ⟩
Column1 → ⟨ longitude, lng ⟩
Column2 → ⟨ latitude, lat ⟩
Output Production:
/ **NL:** Select longitude from circuits sorted by latitude
\SQL: SELECT lng FROM circuits ORDER BY lat

Rule → ⟨ Select Column1 from Table sorted by Column2 Order,
SELECT Column1 FROM Table ORDER BY Column2 Order⟩

Example:
Table → ⟨ debate people , debate_people ⟩
Column1 → ⟨ debate id, debate_id ⟩
Column2 → ⟨ negative, Negativ ⟩
Order → ⟨ in ascending order , ASC ⟩
Output Production:
/ **NL:** Select debate id from debate people sorted by negative in ascending order
\SQL: SELECT Debate_ID FROM debate_people ORDER BY Negative ASC

HAVING

Rule → ⟨ Select Column1 from Table grouped by Column1 with Degree imum Column2 equal to ColumnValue,
SELECT Column1 FROM Table GROUP BY Column1 HAVING Degree Column2 = ColumnValue ⟩

Example:
Table → ⟨ climber , climber ⟩
Column1 → ⟨ name, Name ⟩
Column2 → ⟨ points, Points ⟩
Degree → ⟨ min , MIN ⟩
ColumnValue → ⟨ 6.0, 6.0 ⟩
Output Production:
/ **NL:** Select name from climber grouped by name with minimum points equal to 6.0
\SQL: SELECT Name FROM climber GROUP BY Name HAVING MIN(Points) = 6.0

Figure 2: Example SCFG Production Rules for selected SQL Clauses

MIN

Rule → ⟨ Select minimum Column from Table, SELECT MIN(Column) FROM Table⟩

Example:
Table → ⟨ student addresses , Student_Addresses ⟩
Column → ⟨ monthly rental , monthly_rental ⟩
Output Production:
/ **NL:** Select minimum monthly rental from student addresses
\SQL: SELECT MIN(monthly_rental) FROM Student_Addresses

<=, <,>,>=

Rule → ⟨ Select Column1 from Table when Column2 Equality ColumnValue,
SELECT Column1 FROM Table WHERE Column2 Equality ColumnValue⟩

Example:
Table → ⟨ faculty , faculty ⟩
Column1 → ⟨ faculty, Faculty ⟩
Column2 → ⟨ campus, Campus ⟩
Equality → ⟨ greater than, > ⟩
ColumnValue → ⟨ 20 , 20 ⟩
Output Production:
/ **NL:** Select faculty from faculty when campus is greater than 20
\SQL: SELECT Faculty FROM faculty WHERE Campus > 20

AND

Rule → ⟨ Base ConjunctionPhrase ConjunctionPhrase ColEqualityValue, Base ConjunctionPhrase ConjunctionPhrase ColEqualityValue⟩

Example:
Base → ⟨ Select all columns from parties in events when, SELECT * FROM Parties_in_Events WHERE ⟩
ConjunctionPhrase → ⟨ ColEqualityValue and , ColEqualityValue AND ⟩
ColEqualityValue → ⟨ event id equals 9, Event_ID = 9 ⟩
ColEqualityValue → ⟨ role code equals Organizer, Role_Code = 'Organizer' ⟩
ColEqualityValue → ⟨ party id equals 4, Party_ID = 4 ⟩
Output Production:
/ **NL:**
Select all columns from parties in events when event id equals 9 and role code equals Organizer and party id equals 4
\SQL: SELECT * FROM Parties_in_Events WHERE Event_ID = 9 AND Role_Code = 'Organizer' AND Party_ID = 4

Figure 3: Example SCFG Production rules for other selected SQL operators