Netted?! How to Improve the Usefulness of Spider & Co.

Benjamin Hättasch[1], Nadja Geisler[1] and Carsten Binnig

Technical University of Darmstadt (TU Darmstadt), Department of Computer Science, Hochschulstraße 10, 64289 Darmstadt, Germany
[1]Both authors contributed equally to this research

Abstract
Natural language interfaces for databases (NLIDBs) are an intuitive way to access and explore structured data. That makes challenges like Spider (Yale's semantic parsing and text-to-SQL challenge) valuable, as they produce a series of approaches for NL-to-SQL-translation. However, the resulting contributions leave something to be desired. In this paper, we analyze the usefulness of those submissions to the leaderboard for future research. We also present a prototypical implementation called UniverSQL that makes these approaches easier to use in information access systems. We hope that this lowered barrier encourages (future) participants of these challenges to add support for actual usage of their submissions. Finally, we discuss what could be done to improve future benchmarks and shared tasks for (not only) NLIDBs.

1. Introduction

In a world where ever more data is generated, processed, and relied upon, it becomes continually more significant that data is not only accessible to a small group of people. Information can be contained in text, relational databases, knowledge graphs, and many other formats—but users do not want to deal with heterogeneous sources. What they are interested in is accessing information in an easy manner. The borders between structured and unstructured information keep blurring: when using Google for factual questions, infoboxes might show the answer without the need to open a search result. That result might even be wrapped in a generated sentence when voice search was used and nobody cares whether the sentence was extracted from a web page or generated from a database.

On the other hand, there are good reasons why these different ways of storing information exist. Information access methods should leverage the possibilities of each while providing convenient and ideally unified interfaces. With this goal in mind, natural language interfaces emerged as a data retrieval method, leveraging one of our most flexible and intuitive means of communication.

Relational databases are an essential type of information storage. To query them, users require knowledge of the domain, query language (e.g., SQL), and database schema. Contrarily, the vision for natural language interfaces to databases (NLIDBs) encompasses the ability of any user to interactively explore large datasets without help or extensive manual preparation work [1]. As one of the biggest challenges, the application of NLIDBs requires the means to translate natural language (NL) into SQL queries (NL2SQL) (for a recent comprehensive overview of methods and open problems refer to Kim et al. [2]). However, before such NLIDBs can be used as one of many interfaces for information access (i.e., users can enter their information request using arbitrary words and get a correct answer without knowledge about the database), further research is needed.

Contributions We show that current benchmarks, especially the Spider challenge [3] and the related challenges SparC [4] and CoSQL [5] are not sufficient to measure all relevant aspects and support the emergence of ready-to-use NLIDBs. Yet, to foster research not only on NLIDBs but on systems that integrate and use them, we publish an API called UniverSQL[1] to integrate submissions to the challenges into research prototypes and existing systems. Its core functionality is a wrapper implementation to allow the execution of arbitrary queries on pre- or custom-trained models. We additionally provide two sample implementations of this wrapper for existing NL2SQL translators (EditSQL [6] and IRNet [7]). The code is published under an open source license.

Finally, we provide an overview of the advantages and flaws of Spider and other benchmarks and provide ideas on how the evaluation of NLIDBs could advance.

We hope that this research encourages the use of NLIDBs and further development of approaches and benchmarks. Hopefully, this will help make more information accessible to everyone, regardless of their of background.

Outline The rest of the paper is organized as follows: After briefly describing the Spider challenge and its sib-

[1]https://datamanagementlab.github.io/univerSQL/

DESIRES 2021 – 2nd International Conference on Design of Experimental Search & Information REtrieval Systems, September 15–18, 2021, Padua, Italy
benjamin.haettasch@cs.tu-darmstadt.de (B. Hättasch); nadja.geisler@cs.tu-darmstadt.de (N. Geisler); carsten.binnig@cs.tu-darmstadt.de (C. Binnig)
https://benjaminhaettasch.de/ (B. Hättasch); http://www.dm.tu-darmstadt.de/ (N. Geisler)
0000-0001-8949-3611 (B. Hättasch); 0000-0002-5245-6718 (N. Geisler); 0000-0002-2744-7836 (C. Binnig)

lings in Section 2, we analyze how reproducible and usable the submissions to the shared tasks are in Section 3. In Section 4, we present our prototypical implementation UniverSQL that makes more of these systems usable for research. We examine strengths, weaknesses, and possible further developments of benchmarks in Section 5, before providing a brief final summary in Section 6.

2. What are SPIDER, SparC and CoSQL?

The Spider challenge [3] has become one of the standard evaluations for NLIDBs since its publication in 2018. So far, it was cited 217 times and 71 submissions were made to the shared task. The dataset aims to surpass most existing datasets in size by at least one order of magnitude. At the same time it covers a diverse set of simple and complex SQL queries. This provides the necessary basis for data-driven systems to translate joins, nestings etc., and challenges them to do so to achieve good performance on the development and test data splits.

Alongside the dataset, Spider provides a shared task: Since such a dataset is expensive to create, it is not feasible to create one every time the NLIDB is applied to a new database. The authors suggest that this problem is solved by NLIDB systems capable of generalizing to new databases and performing well across domains. This idea is not entirely new: Systems by e.g., Rangel et al. [8] or Wang et al. [9] already attempted to be domain-independent in one way or another. However, Spider is the first dataset of its size, complexity and quality. The split ensures that each database occurs in exactly one set (training, development, and test). This provides a concrete task description and evaluation process, allowing accurate and comparable measurements of success.

Yu et al. [3] also propose a way of categorizing SQL queries with regard to difficulty in the context of the translation task. The concept regards the number of SQL components, selections, and conditions to label a query as easy, medium, hard, or extra hard. A SQL query is estimated to be harder if it contains more SQL keywords, e.g., a query is considered to be hard if it contains nestings, the EXCEPT keyword, or three (or more) columns in the SELECT statements, three (or more) WHERE conditions, and a GROUP BY over two columns. Even more structures or keywords in one query are considered extra hard. The Spider shared task encourages the submission of models to show up in the leaderboard. There are two variants: the original task does not check value accuracy, but there is also a leaderboard for systems that handle/predict values (not just queries with placeholders).

SparC [4] is the multi-turn variant of Spider. It deals with cross-domain semantic parsing in context and is comparable to Spider in size, complexity and databases. However, queries are arranged in user interactions, providing dialogue-like context. Therefore, it is not sufficient to just translate the current NL utterance into SQL but information from previous queries has to be taken into account. Analogous to Spider, SparC features a leaderboard for variants with and without value handling.

CoSQL [5] takes the challenge to the level of a real conversational agent. It consists of both dialogues and annotated SQL queries simulating real-world DB exploration scenarios. Therefore, the system has to maintain a state. CoSQL defines several challenges, the simplest one mainly adds further context to interpret compared to SparC, the other ones cover generation of suitable responses and intention detection/classification.

3. How reproducible and usable are the challenge submissions?

All three challenges (SPIDER, SparC and CoSQL) feature a public leaderboard where different approaches and their scores on the public, development as well as the unpublished test set are listed. In this section, we will investigate the state of the submissions particularly with regard to how reproducible the submissions are and whether they can be used outside of the exact task. An overview of our analysis can be found in Tables 1 and 2 (as in June 2021). We will quickly interpret those results.

SPIDER The leaderboard for the primary Spider task (without value handling) featured 62 entries in June 2021. Some of them are only small variations of the same system, nevertheless, this boils down to 51 different approaches. Yet, only little more than half (36 or 58 %) of those approaches are published in some way, the remaining approaches are anonymous or contain only names of authors or institutions (so far). For 25 submissions, a link to code is provided, yet, some repositories are empty or the link is invalid. In total, 20 approaches (32 %) have at least some code that could be used as starting point for reproduction. Unfortunately, this is not evenly spaced, only for two of the top ten current submissions (and for four of the top twenty) code is provided.

Two approaches deserve special mention: Shi et al. [10] provide a Jupyter Notebook for translation of user-specified queries on custom data[2] and the code of Lin et al. [11] allows interaction with pretrained checkpoints through a command line interface.

SPIDER (with Value Predection) This variation of the task (additionally covering value handling necessary

[2]https://github.com/awslabs/gap-text2sql/blob/main/rat-sql-gap/notebook.ipynb

Table 1
Analysis of the leaderboard entries for Spider (with (+v) and without (-v) value prediction), SparC & CoSQL. We checked how many different approaches are presented, how many of them reference a publication and how often there is code to at least try to reproduce the approach.

	Spider (-v)	Spider (+v)	SparC	CoSQL
Entries	62	7	17	10
Diff. appr.	51	5	15	8
- Publications	36 (58 %)	6 (86 %)	8 (47 %)	9 (90 %)
- Code	20 (32 %)	4 (57 %)	4 (24 %)	5 (50 %)

Table 2
Analysis of the available repositories for the different challenges. We report whether the repositories are empty or contain code, whether checkpoints/pre-trained models are provided for download and whether the usage of this approach on own data/tables is in some way prepared.

	Spider (-v)	Spider (+v)	SparC	CoSQL
Repositories	15	2	4	5
- Empty?	2 (13 %)	0 (0 %)	0 (0 %)	1 (20 %)
- Code?	13 (87 %)	2 (100 %)	3 (75 %)	4 (80 %)
- Checkpoints	9 (60 %)	2 (100 %)	3 (75 %)	2 (40 %)
- Own data?	2 (13 %)	0 (0 %)	0 (0 %)	0 (0 %)

for translating real NL queries) unfortunately received substantially fewer submissions (seven entries for five approaches and all but one with publication). Four approaches, provide code (only two of six publications).

SparC Although this challenge was published just nine months after the Spider challenge, it received considerably fewer submissions so far. The leaderboard for the variant without value handling has 17 entries for 15 different approaches. For less than half of them (47 %) publications are referenced, for 24 % there is code and three submissions provide pre-trained models for download. For the variant with value prediction, there are only two entries, out of which only one references a publication and no code is provided at all. We therefore did not include this variant in Tables 1 and 2.

CoSQL At the time of writing, the challenge was public for around 20 months. There were only baseline implementations or entries without publications for two of the three variants, only one entry included value handling. The main task received ten submissions by eight different approaches with a publication ratio of 90 %. For half of the approaches there is code, but in only two cases checkpoints can be downloaded and there is no preparation for the use of the models outside the evaluation scripts at all.

Overall, we have to conclude that reproducibility of the approaches submitted to the leaderboards of all challenges is at best mediocre, which is in line with problems of the community and especially research in computer science where reproducibility is still a challenge. ACM conferences try to tackle this through reproducibility challenges and badges in the ACM Digital Library.[3] Yet, publishing code and artifacts that allow others to redo the experiments is still optional.

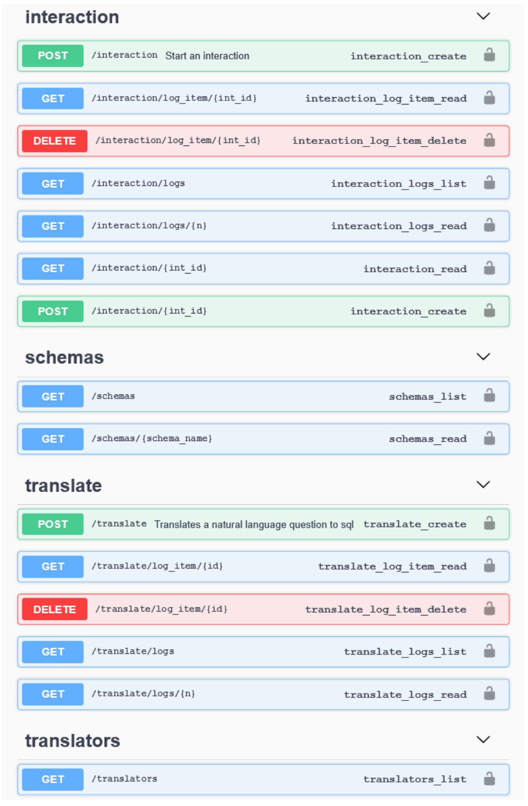
While it is surely not feasible to change the whole publishing and reviewing process at once, we think that shared tasks are a good place to start. Of course it is fine that submissions are anonymous until the approach was reviewed and published. But we advocate that once names are revealed, it should also be necessary to reference publication and code. Authors of a challenge set the requirements for submissions to be included in a leaderboard—and they should take advantage of that.

Moreover, it should be honored when authors of an approach or research prototype invest that extra time to make it directly usable for others and their research.

A very good example (from a slightly different domain) is SentenceBERT [12]. Although it is an implementation accompanying a research paper, it is extremely easy to use: install via pip, import, specify which model to use. The installation scripts will install dependencies and the system will download required files/checkpoints, making it possible to build research on top of it in minutes.

That case is already the cream of the crop, in many cases significantly less effort would help: pinning versions of dependencies (especially machine learning libraries often introduce breaking changes in just months), run the code on a second machine under a different username, add an installation script to download required

[3]https://www.acm.org/publications/policies/artifact-review-and-badging-current



```
interaction
POST    /interaction  Start an interaction                interaction_create
GET     /interaction/log_item/{int_id}                    interaction_log_item_read
DELETE  /interaction/log_item/{int_id}                    interaction_log_item_delete
GET     /interaction/logs                                 interaction_logs_list
GET     /interaction/logs/{n}                             interaction_logs_read
GET     /interaction/{int_id}                             interaction_read
POST    /interaction/{int_id}                             interaction_create

schemas
GET     /schemas                                          schemas_list
GET     /schemas/{schema_name}                            schemas_read

translate
POST    /translate  Translates a natural language question to sql  translate_create
GET     /translate/log_item/{id}                          translate_log_item_read
DELETE  /translate/log_item/{id}                          translate_log_item_delete
GET     /translate/logs                                   translate_logs_list
GET     /translate/logs/{n}                               translate_logs_read

translators
GET     /translators                                      translators_list
```

Figure 1: Endpoints of the UniverSQL API

external data or add environment variables for configuration. Each of these steps can make it substantially easier to run foreign code (or your own after a while). It is not about providing perfectly fast and robust industry-grade software for production use—that is something (academic) researchers usually cannot accomplish and also should not spend their time on—but to allow quick prototypical usage to decide whether it is feasible to use an approach in research and maybe investing time in improving it. We therefore argue that shared tasks like Spider should require this in the future for submissions to their leaderboards, and find it a great pity that most of the current submissions are difficult to reproduce and even more difficult to utilize for further research.

4. Does it translate?

As shown in the last section, in June 2021 there were 86 submission in total for Spider and SparC. If one wants to build a system on top of them, currently one has to pick one of the best performing approaches from the leaderboard, obtain the code, install dependencies, download pre-trained models (if any) and then find a way to run the code not on the benchmark data but on individual natural language queries. There has to be a better way. The Spider and SparC challenges do not enforce a certain architecture (i.e., their aim is to foster research on all kinds of approaches to solve the task and not tie it down to e.g., a hyper-parameter optimization for a fixed architecture). This has the downside of making it even harder to use the resulting approaches in other applications. As a community service we therefore provide a simple API implementation called UniverSQL that can be used in prototypes for information access i.e., ones that use NLIDBs (and maybe other components) but do not focus on implementing them. The idea is that this API can be used as a unified interface to NLIDBs regardless of their architecture. This allows researchers to concentrate on their task—and allows them to make use of approaches that would otherwise be difficult to use.

UniverSQL is a small python application that serve as a translation server. The API allows unified access to most important functionalities (select a database, select a translator, do the actual translation) and some convenience and debugging functions like logging. It can be used for individual translations but also for (context preserving) multi-turn interactions as in the SparC challenge. An overview of available endpoints can be seen in Figure 1.

The core of UniverSQL is a wrapper implementation to allow running arbitrary queries on pre-trained models. We provide two sample implementations of this wrapper for systems from the Spider leaderboard: EditSQL [6] and IRNet [7]. It also includes a script to setup these two systems and download required dependencies and model dumps. We publish our code together with an extensive documentation how to create wrappers for other NL2SQL approaches and scripts for simple setup. We hope that this itself evolves into a challenge were researchers provide such a wrapper implementation and installation script for their approach and will therefore maintain a *ready to use* list as part of the published code.[4]

5. What are we (still) missing?

Modern data driven approaches would not be possible without big amounts of data, but curating and annotating it is out of scope for many researchers. Hence, it is not surprising that Spider and SparC, but also other datasets, have strongly advanced research in the field. However, we believe that further advancements are still possible:

We already outlined some flaws of Spider such as a missing focus on reproducibility. Yet, we also want to highlight advantages like the manually annotated and high-quality data, which deservedly currently makes it the most important benchmark for of NL-2-SQL translators.

[4]https://datamanagementlab.github.io/univerSQL/

In addition to Spider, there are other datasets and approaches for benchmarking of NLIDBs, but, like Spider, they have some flaws. We will take a glance at some of them, to outline typical problems:

The WikiSQL Benchmark by Zhong et al. [13] is a large dataset (though smaller than Spider) that also features a leaderboard. Unfortunately, it consists only of a small number of unique query patterns [14] (in fact, half of the questions in the dataset are generated from one single pattern). In particular, it contains neither joins nor nestings. Furthermore, the NL questions are often low quality (i.e., many are grammatically incorrect), some do not have a proper semantic meaning and make little sense when read by humans and some NL questions do not have the same meaning as the associated SQL query.

Utama et al. [15] published *ParaphraseBench*, an approach that tries to measure translation difficulty by dividing queries into classes. The benchmark was manually curated but is quite small and covers only one table.

A recent paper by Gkini et al. [16] tries to benchmark existing translation systems. They focus on system aspects like execution times or resource consumption and not on translation accuracy. However, their analysis leaves some open questions: First, their dataset which is unfortunately not publicly available (yet) appears to be quite small, it consists only of 216 keyword-based and 241 natural language queries. Second, although they cite Spider, they did not include the high-performing approaches from the leaderboard in their evaluation. Overall, this approach does not appear sufficient for an evaluation that takes the user's perspective into account.

Even if we combined all these approaches, the result would still not be the best way to evaluate NLIDBs. Therefore, we will conclude with a brief outlook on what is still needed and what would be possible in this area.

As mentioned before, it would probably boost the usage of the approaches if they allowed for direct/easy use. Enforcing this is not an inherent part of a benchmark but could be done as part of the setup of a shared task.

Much more difficult but probably also even more important is taking the user's perspective into account. One way to do so could be end-to-end benchmarks that do not only evaluate the translation accuracy but the real performance in a data exploration task from input to the output (SparC and especially CoSQL do this to some extend). But there are many other highly interesting questions: We can measure the accuracy of a system like an NLIDB, but what accuracy should we strive for? Are all errors equally bad? Can a slightly wrong translation still be sufficient? What is the influence of a suboptimal translation? Will the user be satisfied by a system with 100 % translation accuracy? Or do they expect something that cannot be accomplished even by perfectly working systems? Answering such questions is hard, it can probably not always be automated and it is difficult to frame the answer as a bunch of numbers. Yet, a framework to asses a system in respect to these kind of questions would help to better decide on which improvements it is worth to focus. We therefore hope that this user perspective will be considered more regularly in computer science research—not as a separate field of research but an integral part to drive research in an direction that is suitable to support humans best in whatever they want to accomplish.

6. Conclusion

In this paper, we analyzed the reproducibility and preparation for use in further research of the submissions to the Spider, SparC and CoSQL challenges. Unfortunately, we found that only for about 40 % of the submissions code is available and for even fewer submissions artifacts like pre-trained model dumps are provided. Additionally, the code is in most cases only capable to do the batch translation of specific data required for the evaluation scripts of the challenges but not prepared for use on other real world data. We therefore presented a prototypical API implementation called UniverSQL that provides a simple interface for NL to SQL translation and boils down the task of adapting an approach for individual translation to implementing a simple wrapper class. The implementation is provided as open source software. Finally, we analyzed further shortcomings of Spider and other benchmarks and advocated for a stronger user perspective when designing similar benchmarks in the future.

Acknowledgments

We thank Janina Hartmann and Sebastian Bremser for their substantial contribution to the implementation.

This work has been supported by the German Federal Ministry of Education and Research as part of the Project Software Campus 2.0 (TUDA), Microproject INTEXPLORE, under grant ZN 01IS17050, by the German Research Foundation as part of the Research Training Group *Adaptive Preparation of Information from Heterogeneous Sources (AIPHES)* under grant No. *GRK 1994/1*, as well as the German Federal Ministry of Education and Research and the State of Hesse through the National High-Performance Computing Program.

This research and development project is/was partly funded by the German Federal Ministry of Education and Research (BMBF) within the "The Future of Value Creation – Research on Production, Services and Work" program (funding number 02L19C150) and managed by the Project Management Agency Karlsruhe (PTKA). The authors are responsible for the content of this publication.

References

[1] R. J. L. John, N. Potti, J. M. Patel, Ava: From data to insights through conversations., in: CIDR, 2017.

[2] H. Kim, B.-H. So, W.-S. Han, H. Lee, Natural language to SQL: Where are we today?, Proceedings of the VLDB Endowment 13 (2020) 1737–1750. doi:10.14778/3401960.3401970.

[3] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, D. Radev, 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, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 3911–3921.

[4] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, I. L. Heyang Er, B. Pang, T. Chen, E. Ji, S. Dixit, D. Proctor, S. Shim, V. Z. Jonathan Kraft, C. Xiong, R. Socher, D. Radev, SParC: Cross-domain semantic parsing in context, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 4511–4523.

[5] T. Yu, R. Zhang, H. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, et al., CoSQL: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases, Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (2019). URL: http://dx.doi.org/10.18653/v1/D19-1204. doi:10.18653/v1/d19-1204.

[6] R. Zhang, T. Yu, H. Y. Er, S. Shim, E. Xue, X. V. Lin, T. Shi, C. Xiong, R. Socher, D. Radev, Editing-based sql query generation for cross-domain context-dependent questions, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Hong Kong, China, 2019, pp. 5338–5349.

[7] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J.-G. Lou, T. Liu, D. Zhang, Towards complex text-to-sql in cross-domain database with intermediate representation, in: Proceeding of the 57th Annual Meeting of the Association for Computational Linguistics (ACL), Association for Computational Linguistics, 2019, p. 4524–4535.

[8] R. A. P. Rangel, O. Joaquín Pérez, B. Juan Javier González, A. Gelbukh, G. Sidorov, M. M. J. Rodríguez, A domain independent natural language interface to databases capable of processing complex queries, in: A. Gelbukh, Á. de Albornoz, H. Terashima-Marín (Eds.), MICAI 2005: Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 833–842.

[9] Y. Wang, J. Berant, P. Liang, 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), Association for Computational Linguistics, Beijing, China, 2015, pp. 1332–1342. URL: https://www.aclweb.org/anthology/P15-1129. doi:10.3115/v1/P15-1129.

[10] P. Shi, P. Ng, Z. Wang, H. Zhu, A. H. Li, J. Wang, C. N. dos Santos, B. Xiang, Learning contextual representations for semantic parsing with generation-augmented pre-training, 2020. arXiv:2012.10309.

[11] X. V. Lin, R. Socher, C. Xiong, Bridging textual and tabular data for cross-domain text-to-sql semantic parsing, 2020. arXiv:2012.12627.

[12] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2019, pp. 3982 – 3992. URL: https://arxiv.org/abs/1908.10084.

[13] V. Zhong, C. Xiong, R. Socher, Seq2sql: Generating structured queries from natural language using reinforcement learning, CoRR abs/1709.00103 (2017).

[14] C. Finegan-Dollak, J. K. Kummerfeld, X. Lin, K. Ramanathan, S. Sadasivam, R. Zhang, D. R. Radev, Improving text-to-sql evaluation methodology, CoRR abs/1806.09029 (2018).

[15] P. Utama, N. Weir, F. Basik, C. Binnig, U. Cetintemel, B. Hättasch, A. Ilkhechi, S. Ramaswamy, A. Usta, An end-to-end neural natural language interface for databases, 2018. arXiv:1804.00401.

[16] O. Gkini, T. Belmpas, G. Koutrika, Y. Ioannidis, An in-depth benchmarking of text-to-sql systems, in: Proceedings of the 2021 International Conference on Management of Data, SIGMOD/PODS '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 632–644. URL: https://doi.org/10.1145/3448016.3452836. doi:10.1145/3448016.3452836.