Natural Language Data Interfaces: From Keyword Search to ChatGPT, are we there yet?

George Katsogiannis-Meimarakis[1], Christos Tsapelas[1] and Georgia Koutrika[1]

[1]*Athena Research Center*

Abstract

Enabling users to query data in a relational database using natural language has long been considered the holy grail of the database community. Towards this direction, there has been an increasing research focus on Natural Language Data Interfaces that allow users to pose queries in natural language and translate these queries to the underlying database query language. Several approaches have emerged especially due to the recent advances in deep neural networks. Despite this blooming, not only these systems are very complicated and difficult to understand, but they have yet to deliver their promise of enabling users to use natural language to access data easily. Hence, they have failed to see widespread adoption. A question naturally arises: is natural language access to data going to be the elusive holy grail of databases? We hope not. With the aim of fostering research on these open issues, in this position paper, we discuss (currently unmet) requirements for effective natural language data exploration and highlight promising research directions. Finally, we describe how to rethink the text-to-SQL problem and how this should be realized as an integral capability of a DBMS for the realization of a system that fully supports natural language queries over data.

Keywords

natural language interfaces, data exploration

1. Introduction

"If we are to satisfy the needs of casual users of databases, we must break the barriers that presently prevent these users from freely employing their native language" (E. F. Codd, 1974) [1]. Enabling users to query data using natural language has long been the "holy grail" of the database community. Research on Natural Language Data Interfaces (NLIDBs) started almost as early as the first DBMS emerged, in the 70's [2]. Early systems enabled keyword searches. They relied on data indexes to find relations that contained the query keywords and on the database schema to join them and return the answer to a query (e.g., [3, 4]). Parsing-based approaches parsed the input question to understand its grammatical structure and them map it to the structure of the desired SQL query (e.g., [5, 6]). Recently, there has been a growing interest in *neural machine translation approaches* for learning natural language data interfaces [7, 8, 9] (see [10] for a recent survey). Advances in NLP, such as the introduction of Transformers [11], has given a boost in the area, while the latest developments such as ChatGPT [12] seem to promise that human-like conversation is more plausible. But are we really close to "talking to our databases" [13]?

Unfortunately, existing efforts, including the latest deep learning based systems, have not seen widespread adoption and have yet to deliver their promise of enabling users use natural language to search data. In this position paper, we delve into the limitations of existing approaches and we discuss lessons learnt. We argue that despite the recent bloom of approaches, these focus on certain aspects of the problem (e.g., improving the translation accuracy over a specific dataset [14]) and largely miss the big picture. We identify important requirements for natural language data interfaces and highlight open challenges and promising research directions.

2. The Inherent Challenges of NL Data Interfaces

For relational data, translating queries from natural language to SQL is known as the *text-to-SQL problem*. The text-to-SQL problem is notoriously hard with challenges arising both from the NL and SQL sides.

A natural language question (NLQ) may be ambiguous, allowing more than one interpretation. Furthermore, a word may have a different meaning in different contexts. For example, *"top"* in *"top scorer"* may mean the one with the highest (total) number of goals, while in *"top movies"*, it may mean the ones with the greatest rating. On the other hand, completely different words or sentences can have the same meaning. For instance, *"How many people live in Amsterdam?"* and *"What is the population of Amsterdam?"* translate to the same SQL query. Dealing with different NL utterances (paraphrasing) is a challenge, as each one may need different handling.

Proceedings of the 6th International Workshop on Big Data Visual Exploration and Analytics co-located with EDBT/ICDT 2023 Joint Conference (March 28-31, 2023), Ioannina, GR

katso@athenarc.gr (G. Katsogiannis-Meimarakis); ctsapelas@athenarc.gr (C. Tsapelas); georgia@athenarc.gr (G. Koutrika)

On the other hand, SQL has strict syntax, which leads to limited expressivity compared to natural language. Furthermore, while a sentence in natural language may contain some mistakes, and still be understood by a human, a SQL query needs to be syntactically and semantically correct in order to be executable over the underlying data. In fact, the above limitations create enormous challenges for translating NL to SQL queries. SQL's strict syntax may lead to cumbersome translations. A relatively simple NL query may map to a complex SQL query. For example, *"Return the movie with the best rating"* maps to a nested SQL. While the original NL query is simple, building the complex SQL query may be a tough call for the system.

All these challenges make the text-to-SQL problem so hard. Not only it is difficult to understand a NL query but it is also difficult to build the correct SQL query. Even similar questions may lead to a different outcome over different databases: one may be translated over one database and the other may not, due to issues such as ambiguity, paraphrasing, and different schemas.

3. Existing Approaches

The "Database way". One category of approaches tackle the text-to-SQL problem as a *mapping problem* [3, 4]: how to map query elements to database elements (tables, columns and values) and then find the desired interconnections of these data elements that capture the user intent. In addition, *parsing-based approaches* parse the input question to understand its grammatical structure, which is then mapped to the desired SQL query [5, 6].

The "Machine Learning way". The other approach is to tackle the text-to-SQL problem as a *language translation problem*, and train a neural network on a large amount of {NL query/SQL} pairs [10]. Originally, these systems ignored the underlying database, and they did not ensure that the generated SQL is syntactically and semantically correct, i.e., executable over this database. To address this problem, recent approaches employ two additional techniques. First, schema linking aims at the discovery of possible mentions of database elements in the NLQ. These discovered schema links, along with the rest of the inputs, are fed into the neural network that is responsible for the translation. Second, output refinement can be applied on a trained model to avoid producing incorrect SQL queries [15].

Limitations and Lessons Learnt. The "Database way" can handle different types of SQL queries and can work on any database. However, existing approaches struggle with more complex and diverse NL queries and cannot easily cope with NL challenges, such as synonyms, paraphrasing and typos [16].

The "Machine Learning way" promises to be more generalizable both in terms of the different types of NL queries the methods can understand as well as the different databases they can work on. There are also important limitations. In practice, all methods focus on limited-scope problems and their accuracy severely degrades with more complex and diverse NL and SQL queries [17]. Furthermore, they depend on training data and cannot cope with unseen databases and queries. For example, Spider [18], a large-scale text-to-SQL dataset that is very popular for training and evaluating text-to-SQL systems, contains queries over 200 relational databases from 138 different domains. However, these are toy databases with simple schemas and small sizes that fail to reflect the characteristics and difficulties of real-world DBs.

Furthermore, models used so far are typically quite complex and large, questioning their practical use in a complex system, like a database engine [1]. Moreover, most ML approaches used support poor and size-limiting input representations that cannot possibly leverage the wealth of database information comprising hundreds of tables and attributes, data values, and queries.

These limitations become highly relevant when applying a text-to-SQL system to an actual database [14] used in a business, research or any other real-world use case. Such databases can pose difficulties not encountered in the datasets used to train and evaluate such systems, for example, a large number of tables and attributes and table and column names that use domain-specific terminology.

4. Going Forward: Requirements and Opportunities

The challenges of the text-to-SQL problem, the aforementioned observations as well as our own experience with working with and evaluating several text-to-SQL systems [10, 16] point to a set of requirements for a NLIDB.

R1. Query expressivity: Using a query language such as SQL, the user knows exactly what queries are possible. In a similar vein, the set of NL and SQL queries that a NLIDB supports should be clearly defined so that a user is aware of the available query capabilities.

R2. Data independence: A NLIDB should support the same query expressivity for different databases. In other words, the same type of NL query should be possible over any database. For example, if the user could ask "what is the average X of Y" in one database, then this type of query should be possible in any other database.

R3. Performance: Allowing the users to express questions in NL should free them from using SQL but also, from how their question will be executed efficiently. The system should transparently find the most efficient way

[1] The training cost as well as the energy consumption [19] of such big models are important concerns.

to answer a NL query, minimizing both the translation overhead and the execution cost of retrieving the results.

R4. Scalability: A NLIDB should be feasible and scalable over any database.

Requirement R1 is important because up to now almost none of the known text-to-SQL systems provides a clearly defined query language or specification of its query capabilities. For the user, it is a trial-and-error process to see what queries can be understood and answered by the system. Is it possible to come up with a query language specification that systems can refer to in order to describe their query expressivity?

Towards R1, a query categorization in the spirit of [16] may be a good starting point. This could enable the creation of appropriate benchmarks for the comparison of the query capabilities of different systems. Even devising an appropriate query categorization and an appropriate benchmark raises several challenges: what categories to choose, what queries should be in each category, which datasets to use. Furthermore, one should take into account SQL equivalence (different SQL queries that return the same results), and NL ambiguity (a NL query may have more than one correct translation over the data). Unfortunately, existing benchmarks fail to address the query expressivity question. For instance, Spider has four very coarse-grained classes of queries.

Requirement R2 complements R1 in saying that the same query expressivity should be supported over any database. This comes naturally with query languages such as SQL. For instance, SPJ queries can be supported over any data. For a NLIDB, that does not hold. Going from one database to another, the same type of queries may not be supported. As we have already pointed out, this is a major concern for deep learning systems. A system trained over Spider will not work over a new domain such as astrophysics or cancer research.

One could build specialized, domain-specific benchmarks for training and evaluating text-to-SQL systems in domains, such as scientific databases. Manually crafting such benchmarks is prohibitive, especially in these kinds of domains. Data augmentation, i.e. automatic benchmark generation, is an open research direction [20]. However, this is where the power of benchmarks as a means to demonstrate query expressivity ends. How does one ensure data independence is a different beast and finding better training datasets is not the solution to the problem. Rethinking the system design is needed instead.

Towards this direction, approaches that have been proposed by the DB community have been shown to be more effective from the data independence perspective, since they rely on the information that the database provides. This potentially points to the need of re-thinking our approach to the text-to-SQL problem. Some parts of the solution may require DB methods to ensure data independence and some other parts may use neural models to generalize system knowledge, for example on the diversity and complexity of NL queries. How would a system that combines such capabilities look like?

Requirement R3 poses a serious challenge. While the state-of-the-art systems are still dealing with "getting the answer right", they are mostly overlooking the "getting the answer fast". Improving translation speed by building efficient methods is necessary. But this may not be enough. Text-to-SQL systems originating from the DB community not only tried to generate correct SQL queries but also optimal in terms of execution speed. Hence, many of them contained logic for generating code that would return the desired results fast. This may be necessary for a NLIDB. The database community could come up with benchmarks that focus on efficiency (not just effectiveness) and allow evaluating systems based on execution time and resource consumption.

R4 highlights the need for realistic solutions. Deep learning text-to-SQL systems typically rely on very complex models, which have been trained and evaluated on toy databases (contained in existing benchmarks). In several cases, it may not be possible to have the required resources to train such enormous models. Furthermore, since these models require that the database schema is given as input, they do not scale well to very large databases, with hundreds of attributes and tables (such as astrophysics and biological data). Instead of focusing on increasing the model complexity aiming at translation accuracy, we need to design solutions that also take into account system efficiency, complexity, and scale.

To further move the needle, we may need to rethink our approach to the problem. The text-to-SQL problem has been seen as a mapping or a language translation problem. This is an oversimplification, and in fact *the text-to-SQL problem comprises (at least) three (connected) problems*: a representation problem (what is asked), a planning problem (how to answer it) and an optimization problem (how to execute it efficiently). By decomposing the problem into its sub-problems, we can focus on each one and find the best solution, either a DB or ML technique or combination of both. We can investigate knowledge representation schemes that can scale well to very large databases. For planning and optimization, we can focus on system efficiency, complexity, and scale.

We also believe that *natural language query capabilities should be implemented closer to the DBMS*. All the data (and information about the data, such as statistics and metadata) as well as data operations are part of the DBMS. Querying data using natural language requires all the knowledge that a DBMS has on the data as well as its processing capabilities (and will considerably enhance all of them in the process).

As the system processes NL queries, it should learn and improve its query capabilities as well. At the same

time, it can leverage this knowledge for learning how to translate SQL queries to NL. To have a fully natural language access to a database, we also need to consider the SQL-to-NL problem, i.e., how the system can generate NL descriptions of SQL queries that it executes. This is useful so that the system can explain the results the user receives or when the NL query leads to several interpretations.

5. Conclusions

In this position paper, we revisit the "holy grail" of databases: natural language interfaces. We evaluate existing works, we highlight their limitations, and we discuss lessons learnt so far. We identify important requirements for a NLIDB and highlight open challenges and promising research directions. To move the needle, we revisit the text-to-SQL problem, and we argue that natural language access should be realized closer to a DBMS rather than as an external system that provides a NL interface to data. Our intention with this paper is to stir the waters and give a flavor of an exciting research territory. The data interfaces of the future will be more human-like.

Acknowledgments

This work has been partially funded by the European Union's Horizon 2020 research and innovation program (grant agreement No 863410).

References

[1] E. F. Codd, Seven steps to rendezvous with the casual user, in: J. W. Klimbie, K. L. Koffeman (Eds.), Data Base Management, Proceeding of the IFIP Working Conference Data Base Management, 1974, 1974, pp. 179–200.

[2] I. Androutsopoulos, G. D. Ritchie, P. Thanisch, Natural language interfaces to databases - an introduction, Natural Language Engineering 1 (1995) 29–81. URL: https://doi.org/10.1017/S135132490000005X. doi:10.1017/S135132490000005X.

[3] V. Hristidis, L. Gravano, Y. Papakonstantinou, Efficient IR-style keyword search over relational databases, in: VLDB, 2003, pp. 850–861.

[4] Y. Luo, X. Lin, W. Wang, X. Zhou, Spark: Top-k keyword query in relational databases, in: ACM SIGMOD, 2007, pp. 115–126.

[5] F. Li, H. V. Jagadish, Constructing an interactive natural language interface for relational databases, PVLDB 8 (2014) 73–84.

[6] N. Yaghmazadeh, Y. Wang, I. Dillig, T. Dillig, Sqlizer: Query synthesis from natural language, PACMPL (2017) 63:1–63:26.

[7] V. Zhong, C. Xiong, R. Socher, Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017. arXiv:1709.00103.

[8] B. Wang, R. Shin, X. Liu, O. Polozov, M. Richardson, Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers, 2020. arXiv:1911.04942.

[9] 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, 2019. arXiv:1905.08205.

[10] G. Katsogiannis-Meimarakis, G. Koutrika, A survey on deep learning approaches for text-to-sql, The VLDB Journal (2023) (????). doi:https://doi.org/10.1007/s00778-022-00776-8.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, 2017. arXiv:1706.03762.

[12] chatgpt, 2023. URL: https://openai.com/blog/chatgpt/.

[13] A. Simitsis, Y. E. Ioannidis, Dbmss should talk back too, in: Fourth Biennial Conference on Innovative Data Systems Research, CIDR 2009, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings, www.cidrdb.org, 2009. URL: http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_119.pdf.

[14] M. Hazoom, V. Malik, B. Bogin, Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data, in: 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021), 2021, pp. 77–87.

[15] C. Wang, K. Tatwawadi, M. Brockschmidt, P.-S. Huang, Y. Mao, O. Polozov, R. Singh, Robust text-to-sql generation with execution-guided decoding, 2018. arXiv:1807.03100.

[16] O. Gkini, T. Belmpas, Y. Ioannidis, G. Koutrika, An in-depth benchmarking of text-to-sql systems, in: SIGMOD Conference, ACM, 2021.

[17] H. Kim, B.-H. So, W.-S. Han, H. Lee, Natural language to sql: Where are we today?, Proc. VLDB Endow. 13 (2020) 1737–1750.

[18] 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, 2019. arXiv:1809.08887.

[19] O. Sharir, B. Peleg, Y. Shoham, The cost of training NLP models: A concise overview, CoRR abs/2004.08900 (2020). URL: https://arxiv.org/abs/2004.08900. arXiv:2004.08900.

[20] N. Weir, P. Utama, A. Galakatos, A. Crotty, A. Ilkhechi, S. Ramaswamy, R. Bhushan, N. Geisler,

B. Hättasch, S. Eger, U. Çetintemel, C. Binnig, Dbpal: A fully pluggable NL2SQL training pipeline, in: Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference, ACM, 2020, pp. 2347–2361.