NL2INTERFACE: Interactive Visualization Interface Generation from Natural Language Queries

Yiru Chen*
Columbia University

Ryan Li†
University of Washington

Austin Mac ‡
University of California, Santa Barbara

Tianbao Xie§
University of Hong Kong

Tao Yu¶
University of Hong Kong

Eugene Wu‖
Columbia University

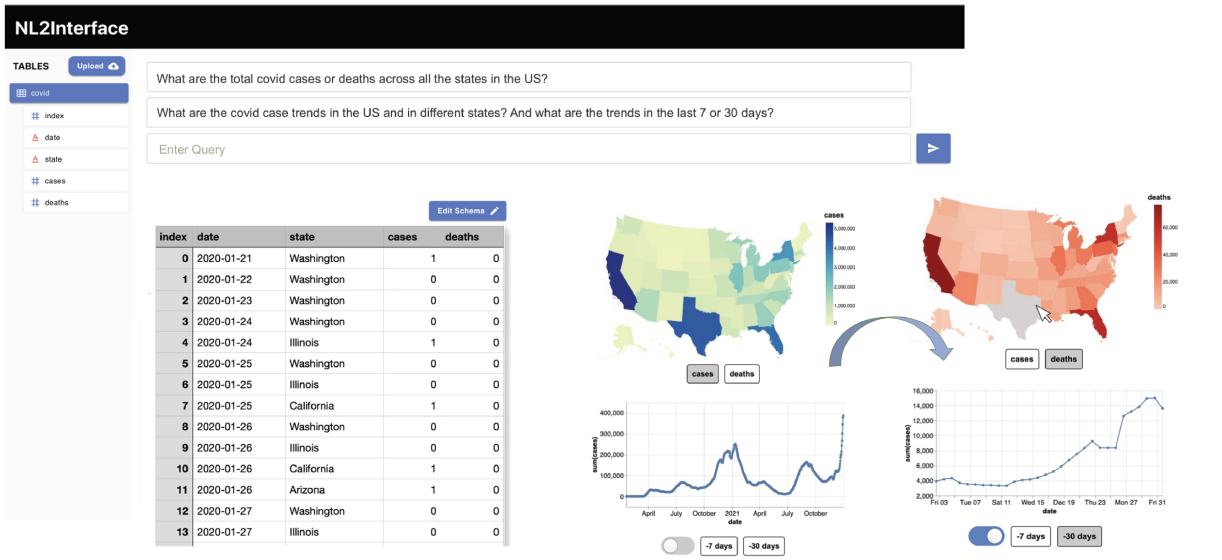

Figure 1: This demonstrates the NL2INTERFACE user interface. A user uploads the COVID-19 dataset using the left panel and the table contents are displayed in the bottom left of the interface. Then, this user types two natural language queries in the top textbox. Our NL2INTERFACE will return an interactive interface consisting of two visualizations, one toggle, two button sets, and a click interaction over the map visualization. The user can interact with the interface to study the covid death trend within the last 30 days in Texas by clicking the state of "Texas" on the map, clicking the "deaths" button, and then clicking the "-30 days" button. The interface after these interactions is shown on the right.

ABSTRACT

We develop NL2INTERFACE to explore the potential of generating usable interactive multi-visualization interfaces from natural language queries. With NL2INTERFACE, users can directly write natural language queries to automatically generate a fully interactive multi-visualization interface without any extra effort of learning a tool or programming language. Furthermore, users can interact with the interfaces to easily transform data and quickly see results in the visualizations.

1 INTRODUCTION

Interactive visualization interfaces (or simply *interfaces*) play a critical role in nearly every stage of data management—including data cleaning [32], wrangling [15], modeling [12], exploration [4, 19], and communication [8, 14]. A specific data management task is a series of data transformations commonly performed with SQL that gradually converts data into an ideal format. The interface design primarily focuses on visualizations and providing interactions. Interactions including widgets and visualization interactions such as brushing and clicking help users easily specify the desired transformations and let users quickly see results in the visualizations.

Such interfaces require considerable expertise and trial-and-error to design and implement. This is because end users have to determine the data management task, then choose visualizations, interactions, and layout to express the underlying data transformations [18].

Existing works [1, 7, 11, 13, 20, 29] allow users to express data transformations in natural language queries and return data visualizations. Using natural language queries for data transformation helps users express the transformation directly without having to learn a tool or a programming language. This is desirable for normal users, especially non-programmers. However, existing works can only create individual visualizations instead of interactive multi-visualization interfaces. Unlike an individual visualization, creating an interactive multi-visualization interface requires much more consideration on widgets choices, layout design, number of visualizations, whether or not to allow interactions on visualizations, types of interactions, etc.

*e-mail: yiru.chen@columbia.edu
†e-mail: lansong@cs.washington.edu
‡e-mail: austinmac@ucsb.edu
§e-mail: tianbaox@cs.hku.hk
¶e-mail: tyu@cs.hku.hk
‖e-mail: ewu@cs.columbia.edu

All the above considerations make it very hard to generate interfaces from natural language queries. One key challenge includes: *What is the target representation of the data transformations for natural language query input?* One natural language query may specify multiple data transformations. For example, a natural language query - "What is the covid trend in different states?" includes multiple different queries varying the state filter. Thus, a new representation is needed to represent the underlying data transformations of natural language queries. After designing the representation, we still need to *translate natural language queries into such a representation, and generate interfaces from such a representation*.

In this paper, we develop **NL2INTERFACE to explore the potential of generating usable interactive multi-visualization interfaces from natural language queries**. Under the hood, we use SQL as the data transformation language. Adapted from a recent representation in PI2, a SQL-based interface generation system [6, 30], we extend SQL with a parameterized syntax to encode the structural similarities and differences between SQL queries. We name it structurally parameterized SQL (SPS). With advancements in large language models [2, 3, 10, 24, 27], we use a transformer pre-trained language model named Codex [3] to translate natural language to SPS. Codex only needs a small number of natural language queries and SPS examples as prompts to predict the SPS representation. Afterwards, we use PI2 to automatically generate fully interactive interfaces from the codex-predicted SPS.

Below is an end-to-end example:

Example 1 (Covid Analysis) *Figure 1 shows the* NL2INTERFACE *user interface. The left panel allows users to upload a dataset and shows the table schemas. The table contents are shown to the right. Users can type one or multiple natural language queries using the textbox. After clicking the send button,* NL2INTERFACE *returns an interface for users to interact with.*

Here, we walk through how to use NL2INTERFACE *to generate a useful interface using a COVID-19 dataset. First, the user uploads the COVID-19 dataset using the upper left* `upload` *button. Then, they can view the table schema directly below the* `upload` *button and the table contents in the bottom left of the UI. Given this dataset, the user is interested in two natural language queries: "What are the total covid cases or deaths across all the states in the US?", and "What are the covid case trends in the US and in different states? And what are the trends in the last 7 or 30 days?" The user types these queries in the text box at the top of the* NL2INTERFACE *interface and clicks the send button.*

NL2INTERFACE *generates an interactive visualization interface consisting of two visualizations, one toggle, two button sets, and a click interaction in the top map visualization. The map visualization corresponds to the first query. Users can interact with the button widgets in the middle of the interface to choose between cases and deaths. The line chart below shows the overall trend in the US. Users can toggle on to specify the date range (e.g. last 30 days), or users can toggle off to see the trend for all data points. Besides widgets, users can click on the map visualization to filter specific states (e.g. "Texas"). The interface after user interactions is shown on the right where the line chart updates to show covid cases for the last 30 days in Texas.*

As shown above, NL2INTERFACE lets users directly input their natural language queries, and automatically generates a fully interactive multi-visualization interface without any extra effort of learning a tool or programming language.

2 RELATED WORK

2.1 Natural Language Interface for Data Visualization

Natural language interfaces for data visualization have been studied extensively [1, 7, 11, 13, 20, 29]. Unlike NL2INTERFACE which uses natural language dialogues to create multi-visualization interfaces, existing works are focused on using natural language dialogues to create individual visualizations.

Cox et al. [7] is an early work which takes a natural language question, determines the appropriate database query, and presents a visualization to the user. The Articulate system [29] and Articulate2 [1] use a conversational user interface to allow users to verbally describe what they want to see. This description is then translated into sketch, analysis, and manipulation commands. Similarly, DataTone [11] generates an individual visualization for natural language queries. Furthermore, it can manage the ambiguity in natural language input as widgets. Eviza [26] and Evizeon [13] investigate natural language as a means of interacting with existing visualizations. NL4DV [20] is a toolkit that supports prototyping natural language interfaces for visualizations. Given an NL query, NL4DV generates visualizations by parsing the NL query to extract attributes and tasks.

In contrast, NL2INTERFACE creates interactive multi-visualization interfaces which consider visualizations, interactions and layout. None of the existing works have explored this.

2.2 Natural Language to SQL Translation

Translation from natural language to SQL (Text-to-SQL) has been widely studied by the NLP community [21, 23, 24, 27, 31, 34]. Difficulties in text-to-SQL are mainly two-fold: encoding a variety of complex relationships between the user's query and multiple tables, and decoding the SQL with valid representations. Many successful solutions include question-table encoding and constrained decoding.

Recently, Shaw et al. [27] showed that fine-tuning a pre-trained T5 [22] language model could yield competitive results on Spider [33] text-to-SQL benchmark. PICARD [24] constrains the auto-regressive decoder during inference time to improve the quality of SQL generation. More recently, large language models (LLMs) like GPT-3 [2] and Codex [3] have been shown to perform incredibly well in many NLP tasks without any training. [23] demonstrates Codex's near state-of-the-art performance on Spider in a zero-shot setting when prompted with in-context examples.

In this work, NL2INTERFACE uses Codex as the back-end LLM. [28] finds that Codex is able to generalize to unseen target programs with a few in-context demonstration examples. In our case, we prompt Codex to generate our designed SPS representations given natural language inputs by providing a few natural language-SPS pairs.

2.3 Precision Interface

The Precision Interface series of work [5, 6, 35] explores interface generation from queries. PI [36] uses SQL queries as a proxy and generates interactive widgets from a sequence of input queries. At a high level, it identifies structural differences between queries and maps them to widgets. However, it does not consider visualizations, visualization interactions, nor layout. PI2 [6, 30] automatically generates fully interactive multi-visualization interfaces from SQL analysis. PI2 proposes DIFFTREEs as a compact representation encoding the structural similarities and differences between SQL abstract syntax trees and searches for a good interface for the SQL analysis given a cost model. Our SPS representation design is based on DIFFTREEs; however, the difference is that SPS is SQL-like text strings rather than a tree structure.

Compared with PI2, NL2INTERFACE is able to generate interfaces from natural language queries which makes it more accessible to everyone, especially non-programmers.

3 NL2INTERFACE SYSTEM

NL2INTERFACE is built upon OpenAI's Codex [3] and our previous work PI2 [6] – the interface generation system from SQL analyses.

Figure 2 shows the overall pipeline. First, NL2INTERFACE prepares a few examples of translating from natural language queries to structurally parameterized SQL (SPS) representations, to build a proper prompt for in-context learning for Codex. Then, given natural language queries and a database catalog, Codex predicts SPS representations of the natural language queries. Finally, NL2INTERFACE follows the interface mapping procedure in PI2 [6] to generate interfaces based on a predefined yet extensible cost model.

Below, we will introduce the SPS representation, how to use the Codex model to predict SPS, and the interface generation for SPS using Example 1.

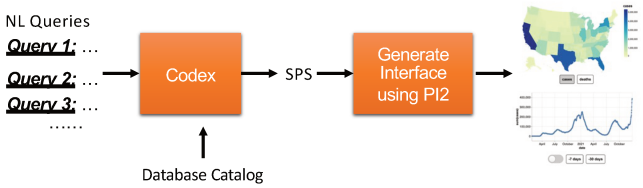

Figure 2: The pipeline for generating interactive interfaces from natural language queries.

3.1 Structurally Parameterized SQL (SPS)

Structurally Parameterized SQL (SPS) is a SQL-like text which extends SQL with choice nodes - `ANY`(choose one of many), `OPT`(exist or not), and `SUBSET`(choose a subset from all choices). It can efficiently represent a large set of queries by extracting and encoding the variations between the queries. SPS's choice nodes can be used to choose from literal parameters, expressions, SQL clauses, or even subqueries. Though SPS is similar to parameterized SQLs [25], the expressiveness of SPS expands from supporting literal parameters as parameterized SQLs [25] to any query expressions, clauses, or subqueries.

Here, we define the choice node syntax:

1. `ANY{c1,..,ck}` can choose one of its k choices between the parentheses, akin to the production rule `ANY→c1|..|ck`. Additionally, to specify a range of numeric values, one can use `ANY{num1-num2}`; e.g., `ANY{3.0-5.0}` represents any numeric value between 3.0 and 5.0. One can also use `&attr` to refer to all distinct values of this attribute in the database. For example, `ANY{&state}` indicates that this `ANY` can choose between all different states in the data.

2. `SUBSET[sep]{c1,..,ck}` represents the production rule `SUBSET→c1?..ck?` with `sep` as the separator. `SUBSET` can choose a subset of its choices listed between the parentheses connected by `sep`. The attribute domain shortcut `&attr` is also applicable for `SUBSET`. For example, `state in (SUBSET[,]{&state})` allows users to choose any set of states in the attribute `state`'s domain.

3. `OPT{t}→t|NULL`. The OPT choice node implies that `t` is optional in the query. This can be viewed as a special case of `ANY` choice node where one choice is empty.

```
q1: select state, sum(cases) from covid group
    by state
q2: select state, sum(deaths) from covid group
    by state
SPS: select state, sum(ANY{cases, deaths}) from
    covid group by state
```

Listing 1: SQL queries and their corresponding SPS for the first natural language query in Example 1.

Example 2 (SPS) *The first natural language query in Example 1 -"What are the total covid cases or deaths across all the states in the US?" contains two queries -* `q1` *and* `q2` *shown in Listing 1. The* SPS *representation shown in Listing 1 uses an* `ANY` *choice node to let users choose between cases and deaths.*

3.2 Codex: Natural Language Queries to SPS

NL2INTERFACE uses Codex to translate natural language queries to SPS representations. Though natural language queries could be arbitrary [37] and even irrelevant to the dataset, we currently focus on natural language queries that correspond to data transformations on the given dataset. We leave detecting inappropriately structured queries and handling unanswerable questions as future work.

Codex is a GPT language model trained on both natural language and publicly available GitHub code [3]. We choose Codex because, given a few examples and without any finetuning, Codex has proven to be comparable to state-of-the-art fine-tuned systems trained on thousands of annotated text-SQL pairs [23], which fits our needs best, since we do not have large-scale training data to re-train a text-to-SPS model.

More specifically, Codex has several model series with different parameter sizes, and we use the *davinci-code-002* model, which is the largest and the most powerful model instance of Codex. Few-shot learning with Codex leverages a recent learning paradigm, in-context learning, and doesn't require any training or fine-tuning. Codex observes a test natural language query and a few text-to-SPS input-output demonstration examples as its input, and directly decodes the corresponding SPS program without any update to its parameters. To format the text-to-SPS input-output demonstration inputs, we follow [23] and provide 5 database schema, each representing a distinct domain: weather, flights, sales, cities, and cars. Under each database schema, we include 10 different natural language to SPS translation examples for that database. The examples are designed to cover the basic usage of `ANY`, `SUBSET`, and `OPT` choice nodes to provide Codex a solid overview of NL to SPS translation. Each example is an <`NL query, SPS`> pair.

Once the user uploads a database and sends a natural language query, we concatenate the schema of the database and the natural language query as the test instance and then append it to the above text-to-SPS input-output demonstration inputs to create a final inference input prompt to Codex. Then the Codex *davinci-code-002* model generates an SPS for each NL query based on the prompt provided.

```
NL Query: What are the covid case trends in the
    US and in different states? And what are the
    trends in the last in 7 or 30 days?

SPS:
  select date, sum(cases) from covid
  where OPT{state = ANY{&state}} and
  OPT{date > date(today(), ANY{'-7 days', '-30 days'})}
  group by date
```

Listing 2: The second natural language query and its corresponding SPS in Example 1.

Example 3 (Codex Prediction) *Listing 2 shows the translated* SPS *for the second NL query in Example 1. In the* SPS, `OPT{state = ANY{&state}}` *corresponds to different states' trend. If* `OPT` *is chosen, it will show a certain state's trend specified by the* `ANY` *choice node.* `&state` *represents all distinct values of attribute* `state` *in the database. Otherwise, it will show the US overall trend.* `OPT {date > date(today(), ANY{'-7 days', '-30 days'})}` *toggles the date range filter. When it is on, the* `ANY` *node will specify the last 7 days or 30 days. Otherwise, it will show the trend for all time.*

In our preliminary work, we evaluate Codex performance by choosing three different databases unseen by Codex and designing four natural language queries for each database. We found that the *davinci-code-002* is able to correctly translate 8 out of the 12 natural language queries into accurate, executable SPS. Further improvements to the prompt and a systematic evaluation of Codex's performance will be conducted in our future work.

3.3 Interface Generation

After translating natural language queries to Structurally Parameterized SQLs (SPS), we follow the interface mapping procedure in PI2 [6] to generate the interface from these SPSs. NL2INTERFACE chooses a visualization for each SPS, chooses widgets or adds visualization interactions to parameterize the choice nodes, and chooses a layout that takes the screen size into account. The visualizations are chosen using a heuristic based on prior work [16] but can use recent visualization recommendations as well. PI2 uses a randomized search procedure and uses a simple yet extensible cost model that takes user effort into account [9].

Example 4 (Interface Generation) *In Example 1, the first* SPS *project attribute shown in Listing 1 is* `state, sum(ANY{cases, deaths})`. *The map visualization renders the query result as a choropleth, while a button below chooses the statistic-total* `cases` *or* `deaths` *to encode as each state's color as shown in Figure 1. The* SPS *in Listing 2's project attributes are* `date, sum(cases)`. NL2INTERFACE *maps this* DIFFTREE *to a line chart in Figure 1. Interestingly, given the first Map visualization,* `OPT{state = ANY{&state}}` *can be directly expressed by the click interaction on the map. Once clicked on the above map, the* `OPT` *node is turned on and the clicked state will parameterize the* `ANY` *choice node. For example, users can click on the "Texas" state on the map and the line chart will show the Texas trend. If the map is not clicked, the* `OPT` *node is turned off and the line chart will show the overall trend in the US. Such a visualization interaction (clicking over the map) has multiple benefits: it makes the interface more concise; it lets users directly interact with the map visualization to specify the transformation; it potentially builds a relation between two visualizations to enable more insights, etc. Finally,* NL2INTERFACE *maps the last part,* `OPT{date > date(today(), ANY{'-7 days', '-30 days'})}`, *to a toggle corresponding to* `OPT` *and a button corresponding to* `ANY`. *Now,* NL2INTERFACE *can return the generated interface to users.*

4 DISCUSSION

NL2INTERFACE is the first to prove the feasibility of automatically generating interfaces from natural language queries. It takes in user specified natural language queries, translates the queries into SPS representations using Codex, and then maps the SPS representations to an interactive multi-visualization interface. However, there are still some limitations.

First, the natural language queries NL2INTERFACE accepts are restricted to those corresponding to data transformation on the given database. However, the NL queries could be arbitrary, irrelevant, or ambiguous [37]. Given recent progress in large pretrained language models, where a single language model is able to achieve state-of-the-art few-shot performance on a wide range of NLP tasks, it is possible to design a pipeline that handles all possible kinds of user queries and reacts correspondingly. For example, Codex can perform a task that detects which queries cannot be mapped to SQL queries given a database and asks the user to ask questions answerable by the database. Besides data transformations, it is also interesting to express the interface preferences in the NL queries, e.g. which visualization to use for a certain NL query to improve the generated interface.

Second, we heavily rely on the prompt design, although Codex is already a powerful model in generalization cross task and domain. Carefully designing the prompt format and examples for forming the prompt is still essential to the performance of prediction [17]. For now, we manually choose examples, make them as diverse as possible at the beginning, and alter them incrementally according to their performance in development in order to maximize performance. The research of automatically selecting proper examples and organizing them a certain way to maximize the ability of Codex is another important direction.

Also, NL2INTERFACE can be used as an interactive human-centered method to allow laypeople to verify the predictions of text-to-SQL neural models such as Codex by checking the resulting visualization. Despite the surprising performance of recent deep learning text-to-SQL methods, interoperability is still one of the key concerns people have with these models. Mapping the predicted target programs to visual interfaces provides users without a technical background a way to verify if these text-to-SQL neural models understand their questions correctly.

ACKNOWLEDGMENTS

Thanks to Alexander Yao, Jeffrey Tao, Brandon Zhang, and Xiaoyue Chen for early contributions. This work was partially supported by NSF 1845638, 1740305, 2008295, 2106197, 2103794, Amazon, Google, and Columbia SIRS.

REFERENCES

[1] J. Aurisano, A. Kumar, A. Gonzalez, J. Leigh, B. DiEugenio, and A. Johnson. Articulate2: Toward a conversational interface for visual data exploration. In *IEEE Visualization*, 2016.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, et al. Language models are few-shot learners, 2020.

[3] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[4] Y. Chen and S. Huang. Tsexplain: Surfacing evolving explanations for time series. *Proceedings of the 2021 International Conference on Management of Data*, 2021.

[5] Y. Chen and E. Wu. Monte carlo tree search for generating interactive data analysis interfaces. In *Intelligent Process Automation Workshop in the AAAI conference*, 2020.

[6] Y. Chen and E. Wu. Pi2: End-to-end interactive visualization interface generation from queries. In *Proceedings of the 2022 International Conference on Management of Data*, pp. 1711–1725, 2022.

[7] K. Cox, R. E. Grinter, S. L. Hibino, L. J. Jagadeesan, and D. Mantilla. A multi-modal natural language interface to an information visualization environment. *International Journal of Speech Technology*, 4(3):297–314, 2001.

[8] FiveThirtyEight. All posts tagged data visualization. https://fivethirtyeight.com/tag/data-visualization/, 2021.

[9] K. Z. Gajos and D. S. Weld. Supple: automatically generating user interfaces. In *IUI '04*, 2004.

[10] Y. Gan, X. Chen, J. Xie, M. Purver, J. R. Woodward, J. Drake, and Q. Zhang. Natural sql: Making sql easier to infer from natural language specifications, 2021. doi: 10.48550/ARXIV.2109.05153

[11] T. Gao, M. Dontcheva, E. Adar, Z. Liu, and K. G. Karahalios. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *Proceedings of the 28th annual acm symposium on user interface software & technology*, pp. 489–500, 2015.

[12] Google. Facets - know your data. https://pair-code.github.io/facets/, 2021.

[13] E. Hoque, V. Setlur, M. Tory, and I. Dykeman. Applying pragmatics principles for interaction with visual analytics. *IEEE transactions on visualization and computer graphics*, 24(1).

[14] iCheck. icheck. icheckuclaim.org, 2021.

[15] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011.

[16] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13, 2007.

[17] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

[18] T. Munzner. *Visualization analysis and design*. CRC press, 2014.

[19] D. Murray. Tableau your data!: Fast and easy visual analysis with tableau software. 2013.

[20] A. Narechania, A. Srinivasan, and J. Stasko. Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):369–379, 2020.

[21] J. Qi, J. Tang, Z. He, X. Wan, C. Zhou, X. Wang, Q. Zhang, and Z. Lin. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *arXiv preprint arXiv:2205.06983*, 2022.

[22] C. Raffel, N. M. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, 2020.

[23] N. Rajkumar, R. Li, and D. Bahdanau. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*, 2022.

[24] T. Scholak, N. Schucher, and D. Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*, 2021.

[25] S. server team. Parameterized query. `https://techcommunity.microsoft.com/t5/sql-server-blog/how-and-why-to-use-parameterized-queries/ba-p/383483`, 2019.

[26] V. Setlur, S. E. Battersby, M. Tory, R. Gossweiler, and A. X. Chang. Eviza: A natural language interface for visual analysis. In *Proceedings of the 29th annual symposium on user interface software and technology*, pp. 365–377, 2016.

[27] P. Shaw, M.-W. Chang, P. Pasupat, and K. Toutanova. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? *arXiv preprint arXiv:2010.12725*, 2020.

[28] R. Shin and B. V. Durme. Few-shot semantic parsing with language models trained on code. In *NAACL*, 2022.

[29] Y. Sun, J. Leigh, A. Johnson, and S. Lee. Articulate: A semi-automated model for translating natural language queries into meaningful visualizations. In *International Symposium on Smart Graphics*, pp. 184–195. Springer, 2010.

[30] J. Tao, Y. Chen, and E. Wu. Demonstration of pi2: Interactive visualization interface generation for sql analysis in notebook. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, p. 2365–2368. Association for Computing Machinery, New York, NY, USA, 2022. doi: 10.1145/3514221.3520153

[31] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson. 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*, pp. 7567–7578. Association for Computational Linguistics, Online, July 2020. doi: 10.18653/v1/2020.acl-main.677

[32] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. 2013.

[33] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

[34] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of EMNLP 2018*. Brussels, Belgium, 2018.

[35] H. Zhang, V. Raj, T. Sellam, and E. Wu. Precision interfaces for different modalities. In *SIGMOD*, 2018.

[36] Q. Zhang, H. Zhang, T. Sellam, and E. Wu. Mining precision interfaces from query logs. In *Proceedings of the 2019 International Conference on Management of Data*, pp. 988–1005, 2019.

[37] Y. Zhang, X. Dong, S. Chang, T. Yu, P. Shi, and R. Zhang. Did you ask a good question? a cross-domain question intention classification benchmark for text-to-sql. *arXiv preprint arXiv:2010.12634*, 2020.