Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks

Yuyu Luo
Tsinghua University, China
luoyy18@mails.tsinghua.edu.cn

Nan Tang
QCRI, HBKU, Qatar
ntang@hbku.edu.qa

Guoliang Li*
Tsinghua University, China
liguoliang@tsinghua.edu.cn

Chengliang Chai
Tsinghua University, China
ccl@tsinghua.edu.cn

Wenbo Li
Tsinghua University, China
li-wb17@mails.tsinghua.edu.cn

Xuedi Qin
Tsinghua University, China
qxd17@mails.tsinghua.edu.cn

ABSTRACT

Natural language (NL) is a promising interaction paradigm for data visualization (VIS). However, there are not any NL to VIS (NL2VIS) *benchmarks* available. Our goal is to provide the first NL2VIS benchmark to enable and push the field of NL2VIS, especially with deep learning technologies.

In this paper, we propose a NL2VIS synthesizer (NL2SQL-to-NL2VIS) that synthesizes NL2VIS benchmarks by piggybacking NL2SQL benchmarks. The intuition is based on the *semantic* connection between SQL queries and VIS queries: SQL queries specify *what* data is needed and VIS queries additionally need to specify *how* to visualize. However, different from SQL that has well-defined syntax, VIS languages (*e.g.,* Vega-Lite, VizQL, ggplot2) are *syntactically* very different. To provide NL2VIS benchmarks that can support many VIS languages, we use a unified intermediate representation, abstract syntax trees (ASTs), for both SQL and VIS queries. We can synthesize multiple VIS trees through adding/deleting nodes to/from an SQL tree. Each VIS tree can then be converted to (any) VIS language. The NL for VIS will be modified based on the NL for SQL to reflect corresponding tree edits.

We produce the first NL2VIS benchmark (NVBENCH), by applying NL2SQL-to-NL2VIS on a popular NL2SQL benchmark Spider, which covers 105 domains, supports seven common types of visualizations, and contains 25,750 (NL, VIS) pairs. Our method reduces the man-hour to 5.7% of developing a NL2VIS benchmark from scratch (or building a NL2VIS benchmark from scratch takes 17.5 × man-hours of our method). Extensive human validation, through 23 experts and 312 crowd workers, demonstrates the high-quality of NVBENCH.

In order to verify that NVBENCH can enable learning-based approaches, we develop a SEQ2VIS model. Our experimental results show that SEQ2VIS works well and significantly outperforms the *state-of-the-art* methods of the NL2VIS task.

ACM Reference Format:
Yuyu Luo, Nan Tang, Guoliang Li*, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, Amsterdam, The Netherlands, 13 pages. https://doi.org/10.1145/3448016.3457261

1 INTRODUCTION

Visualizing data through NL (NL2VIS) is an important step towards democratizing data visualization [31, 32, 47, 55]. Recently, both commercial vendors (*e.g.,* Tableau's Ask Data [50], Microsoft Power BI [2], ThoughtSpot [3]) and academic researchers [11, 42, 46, 52, 62, 70] are starting to explore NL2VIS techniques.

Despite its importance, the study of NL2VIS is still in its infancy (see *e.g.,* the white paper from Thoughtspot [3]). So far, only simple or constrained NL queries are supported. A big obstacle for advancing the field of NL2VIS is the lack of benchmarks, and our goal is to fill this gap. In retrospect, benchmarks have played a key role in spawning the boom in different research communities, such as TPC benchmarks for the database community, ImageNet [12] for image processing, and GLUE [59] and SuperGLUE [58] for NLP.

Challenges. Benchmarking NL2VIS to replicate production-like situations faces three major challenges.

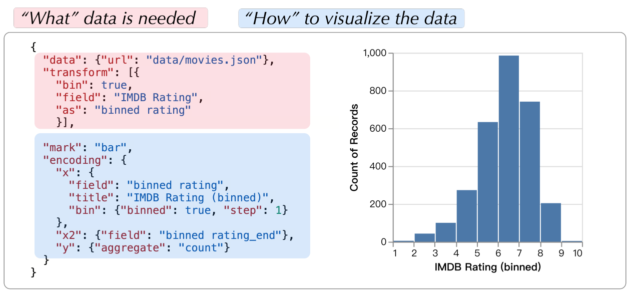
(1) [*Data/Query Coverage.*] Similar to NL2SQL [26], NL2VIS also highly relies on the type of datasets used, because different datasets have different structures, lengths, and various complexities of queries (*e.g.,* selection, comparison, aggregation, join) to be created. Also, it should support the most commonly used charts [5].

(2) [*Diversified* VIS *Languages.*] There are dozens of common VIS languages, *e.g.,* Vega-Lite [48], VizQL [22], ggplot2 [64], ZQL [51], Echarts [27], and many more (see [47] for a survey). Each has its unique syntax: which one(s) to support?

(3) [NL *Variants.*] Users may pose different NL specifications for the same visualization, either *explicitly* (*e.g.,* "draw a pie chart") or *implicitly* (*e.g.,* "show me the proportion").

Common Practice and Its Limitations. The most widely used practice for producing such benchmarks (*e.g.,* Spider [68] for NL2SQL) follows two steps: manually design and collect comprehensive sets of data and queries, and then ask experts to complete tasks. The above approach has two limitations: (1) [*High Human Cost*]: these steps are laborious; even worse, the experts needed are

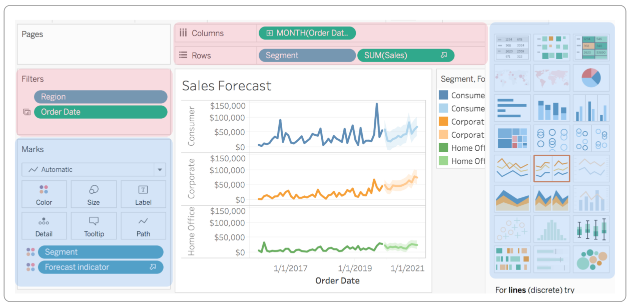
*Guoliang Li is the corresponding author.



```
{
  "data": {"url": "data/movies.json"},
  "transform": [{
    "bin": true,
    "field": "IMDB Rating",
    "as": "binned rating"
    }],

  "mark": "bar",
  "encoding": {
    "x": {
      "field": "binned rating",
      "title": "IMDB Rating (binned)",
      "bin": {"binned": true, "step": 1}
    },
    "x2": {"field": "binned rating_end"},
    "y": {"aggregate": "count"}
  }
}
```

(a) Vega-Lite Visualization Specification



(b) Tableau Desktop Interface

Figure 1: Sample visualization specifications

not always available in a general crowdsourcing environment [39]; and (2) [*Not Extensible*]: for any new dataset, one has to go through the above two steps, from scratch.

The research question we ask in this paper is: *Can we synthesize* NL2VIS *benchmarks that can support* NL2VIS *on many data visualization languages?*

Key Observations. Our key observations are as follows.

(1) [SQL *and* VIS*: Semantic Connection.*] SQL queries ask *what* data is needed (*i.e.,* data operations); and VIS queries additionally just need to specify *how* to visualize the data (*i.e.,* VIS types and other visual encoding details).

EXAMPLE 1. *[*VIS *Queries.] Figure 1 shows two popular ways of specifying visualizations, Vega-Lite [48] and Tableau.*

For Vega-Lite (Figure 1(a)), a user needs to specify what data is needed and how to transform the data (marked in pink), and how to map the data to the visual encoding (marked in blue), in order to produce a bar chart.

Tableau is the *tool for interactive data visualization. As shown in Figure 1(b), a Tableau user also needs to specify a series of data operations (marked in pink) and then choose the right visualization types (marked in blue).*

(2) [SQL *and* VIS*: Syntactic Unification.*] As shown in Figure 2, an SQL query will be represented as an SQL tree, which can be used to synthesize multiple VIS trees. These VIS trees can then be translated to (almost) any target VIS syntax.

(3) [*Reusable Resources.*] We can piggyback NL2SQL benchmarks [68]; and there are remarkable advances in NLP technology [30, 40, 57].

Contributions. We make the following contributions.

(1) [NL2SQL*-to-*NL2VIS *Synthesizer.*] We propose a NL2SQL-to-NL2VIS synthesizer that can synthesize multiple (NL, VIS) pairs from one (NL, SQL) pair. (Section 2)

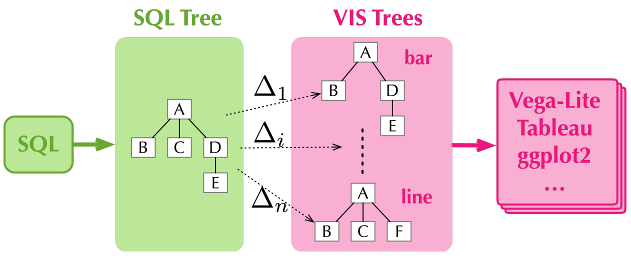


Figure 2: Transforming SQL to multiple VIS languages

(2) [NVBENCH*: Statistics and Evaluation.*] We apply NL2SQL-to-NL2VIS on a popular NL2SQL benchmark Spider [68] to synthesize NVBENCH. NVBENCH contains 780 relational tables from 153 databases in 105 domains (for example, *Sport, College, Hospital*, and so on), and has 7,247 visualizations and 25,750 (NL, VIS) pairs. For evaluation, (i) we invited 23 experts from the NLP, VIS, HCI, DB, and DM communities, all with VIS experience. The results show that 86.9% of synthesized (NL, VIS) pairs are well-matched. In addition, these experts help revise ~2% imperfect NL queries, which results in a refined version, namely NVBENCH*. (ii) We also used crowdsourcing and recruited 312 crowd workers, which shows that 88.7% synthesized (NL, VIS) pairs are good. (Section 3)

(3) [SEQ2VIS*:* NL2VIS *Neural Translation.*] We apply a well known SEQ2SEQ model [7] to learn the translation from NL queries to VIS queries, namely SEQ2VIS. Experiments show that SEQ2VIS achieves ~65% accuracy on NVBENCH, which significantly outperforms the state-of-the-art approaches of the NL2VIS task. (Section 4)

(4) [*Conclusion.*] We propose **(i) the first** NL2VIS **synthesizer** (NL2SQL-to-NL2VIS), and produce **(ii) the first** NL2VIS **benchmark** (NVBENCH). We are also **(iii) the first to apply a neural translation model (**SEQ2VIS**) on learning** NL2VIS and empirically show that it works well. (Section 6)

The code and benchmark is available at **https://github.com/TsinghuaDatabaseGroup/nvBench**, to help both the database community and the visualization community, as well as commercial vendors, to push the field of NL2VIS.

2 NL2SQL-TO-NL2VIS SYNTHESIZER

2.1 Solution Overview

Figure 3 overviews the NL2SQL-to-NL2VIS synthesizer, which consists of two steps: VIS synthesis (*i.e.,* generating visualizations based on SQL queries) and NL synthesis (*i.e.,* editing the NL queries of SQL queries based on the synthesized VIS).

It takes a (NL, SQL) pair (n_Q, Q) as the input, and returns as output a set of (NL, VIS) pairs: $\{(n_{11}, t_1), \ldots, (n_{1k}, t_1), \ldots, (n_{m1}, t_m), \ldots, (n_{mk}, t_m)\}$. Here, Q is an SQL query, t_Q is its (equivalent) tree representation, and n_Q is its corresponding NL query. The output contains m VIS trees $\{t_1, \ldots, t_m\}$; each VIS tree t_i relates to multiple NL queries (*e.g.,* $\{n_{11}, \ldots, n_{1k}\}$ are k NL variants for query t_1).

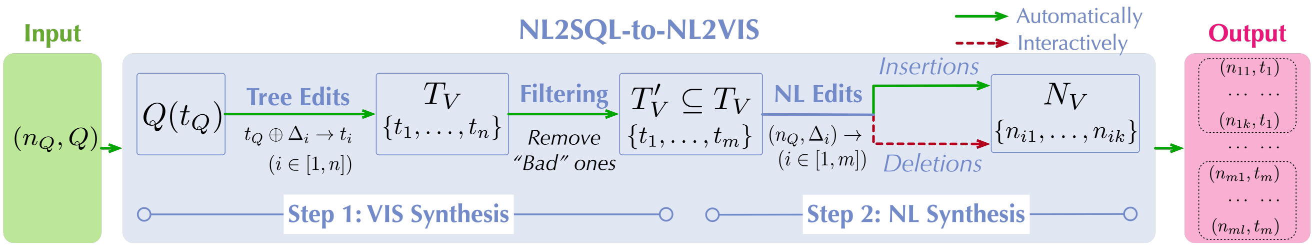


Figure 3: An overview of the NL2SQL-to-NL2VIS synthesizer

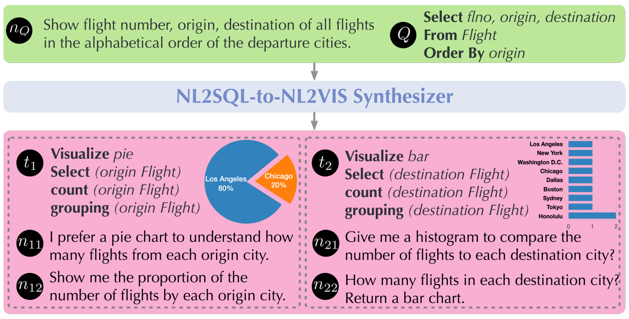

n_Q Show flight number, origin, destination of all flights in the alphabetical order of the departure cities.
Q Select flno, origin, destination From Flight Order By origin
NL2SQL-to-NL2VIS Synthesizer
t_1 Visualize pie Select (origin Flight) count (origin Flight) grouping (origin Flight)
Los Angeles 80%
Chicago 20%
n_{11} I prefer a pie chart to understand how many flights from each origin city.
n_{12} Show me the proportion of the number of flights by each origin city.
t_2 Visualize bar Select (destination Flight) count (destination Flight) grouping (destination Flight)
Los Angeles
New York
Washington D.C.
Chicago
Dallas
Boston
Sydney
Tokyo
Honolulu
n_{21} Give me a histogram to compare the number of flights to each destination city?
n_{22} How many flights in each destination city? Return a bar chart.

Figure 4: A running example for NL2SQL-to-NL2VIS

EXAMPLE 2. *[Input.] Figure 4 gives a real example from a* Flight *dataset, where the original* (NL, SQL) *pair* (n_Q, Q) *from the Spider [68]* NL2SQL *benchmark is given as input.*

[Output.] The NL2SQL*-to-*NL2VIS *will synthesize* T'_V *with two* VIS *queries,* t_1 *and* t_2 *(We will explain the syntax of* t_1 *and* t_2 *in Section 2.2).* t_1 *is a pie chart and* t_2 *is a bar chart. For* t_1*, it synthesizes two* NL *queries* n_{11} *and* n_{12}*; and for* t_2*, it also synthesizes two* NL *queries* n_{21} *and* n_{22}*. Hence, it will output four pairs* $\{(n_{11}, t_1), (n_{12}, t_1), (n_{21}, t_2), (n_{22}, t_2)\}$.

Step 1. VIS **Synthesis.** It does tree edits to obtain multiple VIS trees from one SQL tree, considered as VIS candidates.

[Invariant (*what data*).] An SQL query Q (or equivalently, its tree representation t_Q) is to retrieve *what* data is needed. Because NL2SQL benchmarks have enough coverage and diversity, we try to keep the data querying part unchanged.

[Variant (*how to visualize*).] Tree edits may modify the SQL tree t_Q by adding *how* to visualize (*i.e.,* the VIS type) and some VIS related data operations (*e.g.,* grouping and binning), as well as deleting some nodes (*e.g.,* an SQL query may select more attributes than needed for VIS).

This will result in a set $T_V = \{t_1, \ldots, t_n\}$ of VIS trees. We denote by Δ_i the tree edits on SQL tree t_Q to get VIS tree t_i.

In order to ensure that each VIS query *w.r.t.* a VIS tree in T_V is "good", *e.g.,* a bar chart with several hundred bars is not readable thus is bad. Hence, we need to filter "bad" charts, while only keeping good charts as $T'_V = \{t_1, \ldots, t_m\}$.

Step 2. NL **Synthesis.** Given the input (n_Q, t_Q), each good VIS query t_i, and the tree operations Δ_i that convert t_Q into t_i, it will revise n_Q to reflect the change of Δ_i, and get variants of NL specifications. The purpose of having variants of NL specifications, which is a way of data augmentation [18], is to train a robust model.

Root ::= Q | Visualize Q
Q ::= intersect R R | union R R| except R R | R
R ::= Select | Select Filter | Select Order | Select Superlative
| Select Order Filter | Select Superlative Filter | Select Group
| Select Group Filter | Select Group Order | Select Group Superlative
| Select Group Order Filter | Select Group SuperlativeFilter
Visualize ::= bar | pie | line | scatter | stacked bar | grouping line
| grouping scatter
Select ::= A | A A | A A A | A ... A
Order ::= asc A | desc A
Superlative ::= most V A | least V A
Group ::= grouping A | binning A
Filter ::= and Filter Filter | or Filter Filter | > A V | > A R | < A V | < A R
| >= A V | >= A R | <= A V | <= A R | != A V | != A R | = A V | = A R
| between A V V | between A R | like A V | not like A V
| in A R | not in A R
A ::= max C T | min C T | count C T | sum C T | avg C T | C T
C ::= column
T ::= table
V ::= value

Figure 5: The grammar for (SQL and VIS) AST

At the end, it will produce a set of (NL, VIS) pairs as: $\{(n_{11}, t_1), \ldots, (n_{1k}, t_1), \ldots, (n_{m1}, t_m), \ldots, (n_{ml}, t_m)\}$.

2.2 Bridging SQL and VIS Queries with AST

An ideal grammar to bridge SQL and VIS queries is desired to be: (1) *uniform:* it can represent both SQL and VIS queries; (2) *language-agnostic:* it can be converted to either an SQL query or a VIS query with a specific language (*e.g.,* Vega-Lite); and (3) *extensible:* it can be extended to support other SQL queries or more visualization types.

One choice is based on Abstract Syntax Tree (AST). In particular, we extend SemQL [21], which was used for NL2SQL, to further support NL2VIS. The extended grammar is shown in Figure 5.

[SQL *Scope.*] It supports SQL queries with "select" to project on attributes, "from" which tables as *A* ::= *C* (*column*) *T* (*table*), "where" and "having" in the *Filter* conditions, "order by" through *Order*, "group by" via *grouping A*, "LIMIT" using *Superlative*, "aggregations" on attributes such as {max, min, count, sum, avg}, and combing results such as {intersect, union, except}.

[VIS *Scope.*] It supports seven types of commonly used visualizations [5], namely {bar, pie, line, scatter, stacked bar, grouping line, grouping scatter}. Also, the "binning" operation is supported using *binning A*.

Figure 6 shows an SQL AST tree t_Q and a VIS AST tree t_1, relative to the running example in Figure 4.

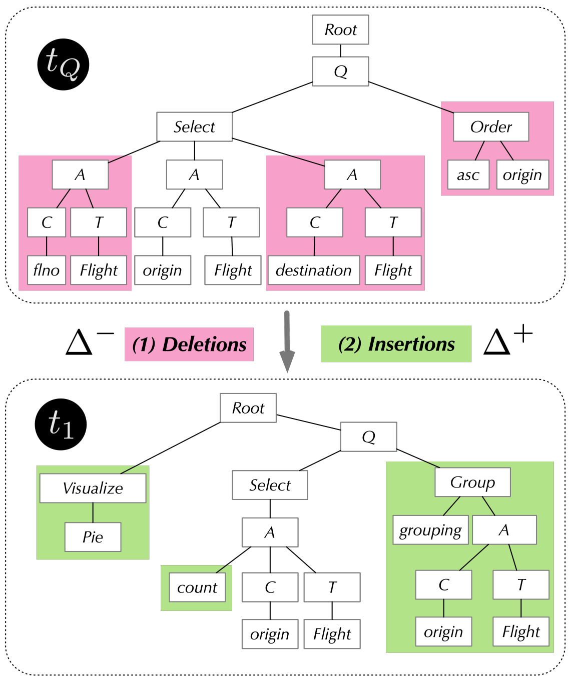


Figure 6: Sample SQL and VIS trees

2.3 Tree Edits: Generating VIS Candidates

Next, we discuss how to generate (candidate) VIS trees from one SQL tree. Intuitively, given one SQL tree, we can "delete" any tree nodes or "insert" any tree nodes, as long as we can produce a valid VIS tree rooted with "$Visualize\ Q$" (Figure 5).

Enumerating all valid VIS trees given a well-defined grammar is not hard (see *e.g.,* [35, 41]). The question is: *For the purpose of* NL2VIS*, do we need to enumerate them?* Just like NL2SQL, do we need to have all SQL queries? The answer is clearly "*no*" because if many queries have similar tree structures, their corresponding NL queries will also be similar – providing many similar, or redundant (NL, VIS) pairs, is not helpful from the perspective of benchmarking NL2VIS.

Moreover, existing NL2SQL benchmarks have paid a lot of efforts on designing SQL queries with different complexities for *what data is needed.* Naturally, if we piggyback NL2SQL benchmarks, we should keep their *what data* part, but focus on adding *how to visualize.*

Candidate VIS Generation. Given an SQL tree t_Q, we first perform different *deletions* on t_Q to get a set $\mathbf{I}$ of intermediate SQL trees $\{t_1^{\mathbf{I}}, \ldots, t_l^{\mathbf{I}}\}$. For each intermediate SQL tree $t_i^{\mathbf{I}} \in \mathbf{I}$, we then make *insertions* to get a set of VIS trees $T_V = \{t_1, \ldots, t_n\}$. The tree edit from t_Q to t_i ($i \in [1, n]$) is denoted by Δ_i, consisting of deletions (Δ_i^-) and insertions (Δ_i^+).

[Deletions (Δ^-).] An SQL tree mainly contains five types of subtrees, *i.e., Select, Order, Filter, Superlative,* and *grouping A*. We keep the last three unchanged (which can be mapped to VIS languages), and only edit the first two, *Select* and *Order*.

(*Select.*) An SQL query may select many attributes but a VIS query typically needs one attribute (*e.g.,* for pie charts), two attributes (*e.g.,* for line charts), or three attributes (*e.g.,* for stacked bar charts). Hence, given the set of attributes in t_Q, we will enumerate and keep all single attributes, the combination of two attributes, and the combination of three attributes – each will result in an intermediate SQL tree in $\mathbf{I}$.

(*Order.*) If *Order* is present in an SQL tree $t^{\mathbf{I}} \in \mathbf{I}$, we will keep $t^{\mathbf{I}}$ in $\mathbf{I}$, and insert another tree $t^{\mathbf{I}'}$ into $\mathbf{I}$ by deleting the *Order* subtree, because *Order* may not be needed for some visualizations (*e.g.,* no orders in pie charts).

[Insertions (Δ^+).] Next we discuss how to do insertions for each $t^{\mathbf{I}} \in \mathbf{I}$ and output a set T_V of VIS trees.

(*Group.*) We can either perform grouping or binning on one attribute *A*. If grouping/binning is present, we add an aggregate (*e.g., max, min, avg*) node. For binning, if the column is temporal, we bin the values by minute, hour, day of the week, month, quarter, or year; if the column is numeric, we follow the convention by setting $binSize = \lceil \frac{maxValue - minValue}{\#bins} \rceil$ (by default, $\#bins = 10$).

(*Visualize.*) We will add a *Visualize* subtree. Also, we need to ensure that the VIS type (*e.g., bar, line*) added can lead to a valid VIS. We follow the rule of thumb (see Table 1) of VIS *w.r.t.* attribute types from the data visualization community [38, 65], which are encoded as rules in our system.

(*Order.*) Order operation can be applied on *bar, stacked bar, line, or grouping line charts* to sort the x-axis or y-axis.

After different combinations of insertions above, we will obtain candidate VIS set T_V.

Example 3. *[Tree Edits.] Figure 6 shows how to convert the* SQL *tree t_Q to one* VIS *tree t_1, by first deleting attribute A-subtrees for* flno *and* destination*, and the* Order *subtree, followed by inserting the* count *predicate, and two subtrees, one for* Group*, and the other for* Visualize.

Table 1: Rules for syntactically correct charts

C (Categorical), T (Temporal), Q (Quantitative)

One Variable	
C	*grouping* + *count* → {bar, pie}
T	*grouping/binning* + *count* → {bar, pie, line}
Two Variables	
C + Q	*grouping/binning/none* + *agg* → {bar, pie}
T + Q	*grouping/binning/none* + *agg* → {bar, pie, line}
Q + Q	scatter
Three Variables	
T+Q+C	*grouping* + *binning* + *agg* → {grouping line, stacked bar}
C+Q+C	*grouping(s)* + *agg* → {stacked bar}
Q+Q+C	*grouping(s)* + *agg* → {grouping scatter}

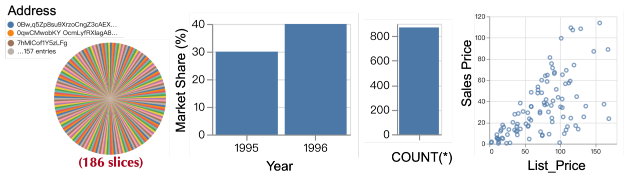
2.4 Filtering Bad Visualizations

We use a pre-trained ML model $\mathbf{M}()$, DeepEye [35][1], to prune candidate VIS queries that are bad. Given a VIS query v, $\mathbf{M}(v)$ outputs either true (*i.e.,* a good VIS) or false (*i.e.,* a bad VIS).

By doing so, we prune bad visualizations from the candidate VIS set T_V and get a set of good VIS set T'_V.

Working Mechanism of DeepEye. It first uses expert rules (rules-of-thumb from the visualization community) to remove invalid visualizations (*e.g.,* one cannot visualize a line chart with two categorical values) and obviously bad visualizations. It then uses a

[1]https://github.com/Thanksyy/DeepEye-APIs



(a) TPC-H ✗ (b) TPC-H ✓ (c) TPC-DS ✗ (d) TPC-DS ✓

Figure 7: Sample visualizations from TPC-H/TPC-DS

trained a binary classifier to decide whether a VIS is good or bad. The binary classifier was trained using 2520/30892 labeled good/bad charts, using features such as the number of distinct values, the number of tuples, the ratio of unique values, max and min values, data type, attribute correlations, and VIS type.

We also use TPC-H (http://www.tpc.org/tpch/) and TPC-DS (http://www.tpc.org/tpcds/) benchmarks to test our VIS queries transformation and filtering mechanisms. We find that below types of VIS queries are filtered out by DeepEye as bad visualizations: (1) *a single value:* this type of query retrieves a single value from the database, and it is not suitable for visualization; (2) pie charts having many slices, (3) bar charts with too many categories, and (4) line charts with two qualitative variables.

EXAMPLE 4. *Figure 7 shows four sample visualizations, where (a) and (b) are transformed from the TPC-H Q_{20} and Q_8, respectively, and (c) and (d) are transformed from TPC-DS Q_9 and Q_7, respectively.*

Figures 7 (a) and (c) are bad visualizations and will be filtered out by DeepEye, because (a) shows too many slices and is hard to derive useful insights from such a pie chart, and (c) is a bar chart to show a single value, which is better to display simply with a table. Figures 7 (b) and (d) are good visualizations: (b) is a bar chart that shows the trend of market share changes over years, and (d) is a scatter chart that shows the correlation between two variables.

2.5 NL Edits: Generating NL Variants

Each VIS $t_i \in T'_V$ is associated with a set of tree edits Δ_i, *i.e.,* deletions Δ_i^- and insertions Δ_i^+. Next, we need to modify the NL of the corresponding SQL query to reflect these changes.

Insertions. For insertions such as *grouping, aggregation, Order,* and VIS types, we use NL extracted from Tableau's Ask Data [50] and NL4DV[2] as rules to enrich the text. For *binning*, we conduct a user study to collect use cases of how users describe binning operations using NL. The table below shows some sample NL rules for inserting different tree nodes.

Visualize: "*{plot, visualize, show, give, draw,...} as a* VIS *type*"
Order: "*{order, sort, list, ...} by A (in {asc, desc, alpha} order)*"
Agg: "*{count, sum, average, how many,...} A (by grouping)* "
grouping: "*by (each {item, category, ...} in) A*"
binning: " *{with, in, ...} a bin/bucket of {month, year, ...}*"

EXAMPLE 5. *Consider an* NL *query that asks "how many male and female faculties do we have?" for the* SQL *query "SELECT COUNT(*) FROM Faculty GROUP BY sex".*

Assume that the VIS *query is simply to visualize the above result using a pie chart,* i.e., *adding "VISUALIZE pie" to the corresponding* SQL *tree. The* NL *edits will automatically revise the above* NL *for* SQL *to a* NL *for* VIS *as "***show the proportion about** *how many male and female faculties do we have?"*

When using rule-based NL insertions to reflect Δ^+, one drawback is that the obtained NL specification may not look very natural. We adopt a popular NLP technique called *back-translation* [16] to smooth it, *e.g.,* first translating an English sentence to French, and then translating it back to English. All the NL specifications in our benchmarks are smoothed using back-translation.

Deletions. It is hard for us to automatically rewrite the origin NL based on the deletions on NL AST tree. The first reason is that some descriptions of SQL clauses may be implicitly given in NL, so it is difficult to match the deleted tree nodes with the corresponding NL phrases. The second reason is that it is difficult to ensure that the semantics and grammar of the NL are correct after deleting the corresponding description. For these deletions, we manually revised the NL queries.

2.6 From VIS Trees to Visualizations

The last step is to convert a VIS tree to a targeted VIS language, so as to be rendered. The mapping from a VIS tree to a targeted VIS language is hard-coded. Currently, we support both Vega-Lite and ECharts, with ~240 and ~320 lines of Python code, respectively. There are also tools that translate among different visualizations, *e.g.,* from ggplot2 to Vega-Lite (https://github.com/vegawidget/ggvega). We also plan to support other popular VIS languages.

3 NVBENCH: STATS AND EVALUATION

3.1 Setup

We use NL2SQL-to-NL2VIS on Spider [68], the latest, popular, large-scale, complex and cross-domain, and with the most challenging NL2SQL tasks (see Section 5 for more discussion).

NL2SQL **Benchmark.** The Spider benchmark consists of 200 databases (averagely 5.1 tables for each database) on 138 domains (for example, *college, club, TV show, government*, and so forth), and 10,181 (NL, SQL) pairs on these databases. These (NL, SQL) pairs were designed to well cover different domains, and various difficulties of NL2SQL tasks [68].

Applying NL2SQL**-to-**NL2VIS **Synthesizer.** When applying NL2SQL-to-NL2VIS on these (NL, SQL) pairs, some may not contribute to any meaningful VIS queries. These meaningless VIS queries will be pruned (see Section 2.4), and their corresponding tables or databases will not be incorporated in our NVBENCH statistics. We have 1,838 VIS objects (25.36% of total cases) that need experts to perform NL edits due to tree deletions, which were completed by two PhD students. For editing one NL query *w.r.t.* tree deletions, the PhD students spent ~1 minute on average. Finally, they produced 3,500 NL variants (13.59% of total NL variants) for the 1,838 VIS objects. The total time is estimated as 3500*1/60/24 = ~2.4 days.

[2]https://nl4dv.github.io/nl4dv/showcase.html

Table 2: nvBench dataset statistics

Coverage		
#-Databases	#-Tables	#-Domains
153	780	105

Top-5 Domains (#-Tables)				
Sport (62)	Customer (52)	School (41)	Shop (35)	Student (22)

The Numbers of Columns and Rows			
#-Cols	Avg (#-Cols)	Max (#-Cols)	Min (#-Cols)
4,017	5.26	48	2
#-Rows	Avg (#-Rows)	Max (#-Rows)	Min (#-Rows)
1,000,572	1309.65	183,978	1

The Types of Columns		
C (%)	T (%)	Q (%)
2763 (68.78%)	465 (11.58%)	789 (19.64%)

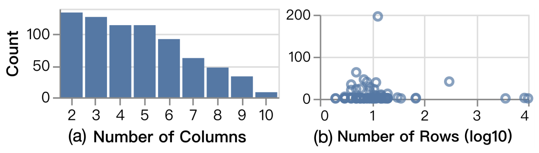


Figure 8: Statistics of Columns and Rows

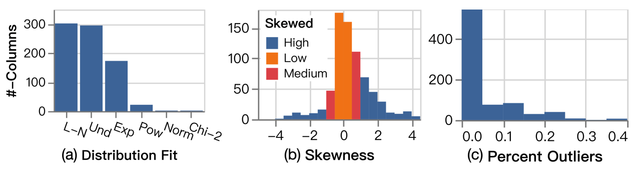


Figure 9: Column Level Statistics

3.2 Statistics of nvBench

Datasets. Table 2 summarizes the statistics of nvBench: the coverage of databases/tables/domain, the numbers of tuples and columns, and the types of columns.

Coverage. nvBench contains 153 databases with 780 tables in total, and covers 105 domains. The top-5 domains and the numbers of tables they contain are also shown, *e.g.*, domain *Sport* has 62 tables, *Customer* has 52 tables, and so on.

The Numbers of Columns and Rows. The total number of columns (resp. rows) of the 780 tables is 4,017 (resp. 1,000,572), with an average of ~5 columns (resp. ~1,309 rows) per table (Table 2). The maximum/minimum number of columns (resp. rows) is 48/2 (resp. 183,978/1).

The Types of Columns. We classify the column type as either categorical (C), temporal (T), or quantitative (Q). Categorical and quantitative columns account for 88.42% of all columns, where categorical type takes 68.78%. The temporal columns account for 11.58% of all columns (Table 2).

Figure 8 provides details about the distributions of columns and rows. Figure 8(a) shows that most tables have less than 5 columns. Figure 8(b) shows that most tables are small, with 5 to 100 rows.

Besides, for quantitative columns, we also tested the goodness-of-fix of six well-known distributions: the normal (abbr. Norm), log-normal (abbr. L-N), exponential (abbr. Exp), power-law (abbr. Pow), uniform (abbr. Unif), and chi-square (abbr. Chi-2) distributions. As shown in Figure 9(a), the most common type of distribution is log-normal, with 302 columns. Note that 295 columns do not follow the above six distributions and there is no column follows the uniform distribution. Only 2 columns with chi-square distribution, which accounts for the least proportion. Figure 9(b) reports the feature of data skewness, which shows that about 335 (42.46%) columns with approximately symmetric distributions, 158 (20.03%) columns with moderately skewed distributions, and ~40% of the columns have a highly skewed distribution. We then detect the percent of outliers in a quantitative column, we say a data point is an outlier if it is more than 1.5× interquartile ranges (IQRs) above the third quartile or the first quartile. Figure 9(c) shows that about 468 (59.31%) quantitative columns do not contain outliers, and around 174 (22.05%) columns have 1%-10% outliers.

vis and nl Queries. nvBench consists of 25,750 (nl, vis) pairs and 7,247 distinct visualizations (Table 3). Next we discuss more details.

vis *Types.* nvBench consists of 7,247 distinct visualizations, including seven types: bar (histogram), pie, line, scatter, stacked bar, grouping line, and grouping scatter charts.

Among them, the number of (stacked) bar charts is the largest, accounting for nearly 80%, while the quantity of (grouping) scatter chart is the smallest because the amount of quantitative columns is limited.

Bar charts and its variants take a huge percentage matches real-world cases. As shown by Beagle [5] and SEEDB [56], bars (and histograms) are the most popular vis types.

vis *Hardness.* Intuitively, not all nl2vis tasks are equivalently hard. Along the same line of Spider [68] for nl2sql, we categorize vis queries into 4 levels: *easy, medium, hard*, and *extra hard*, based on the complexity of the vis tree.

We first define three classes of AST subtrees: (S1) {*Select*, *Order*, *Group*, *Filter*, *Superlative*}, (S2) {the number of A-subtrees, *Filter*-subtrees, and *Group*-subtrees no more than two, respectively}, and (S3) {*intersect*, *union*, *except*}. Next we further define some rules for a vis tree: (R1) it satisfies no more than two conditions in the set of S_2; (R2) it has two subtrees from S_1 in total and meets no more than one rule of S_2; (R3) it satisfies at least three rules from S_2, less than three subtrees from S_1 and without keywords from the set of S_3; (R4) it has three subtrees from S_1 and meets less than three rules from S_2 and without keywords from S_3; and (R5) it has no more than one subtree from S_1 and meets no rule in S_2, but exactly has one keyword from S_3.

Next we define the hardness of vis tree t.

- *Easy:* t has are no more than one subtree from S_1, and the number of A-subtrees no more than two.
- *Medium:* t satisfies either R1 or R2.
- *Hard:* t satisfies either R3, R4, or R5.
- *Extra Hard:* t has more conditions than the hard case.

Figure 10 shows the distribution of synthesized visualizations in different types and hardness. We can see that most visualizations are medium, which account for 38.64%. Also, (stacked) bar charts are the most popular.

nl *Queries.* Table 3 also shows the number of (nl, vis) pairs, the lengths of these nl queries, and the diversity of nl queries.

Table 3: nvBench NL queries and VIS queries

VIS Types	#-VIS	#-(NL, VIS)	#-(NL, VIS)/#-VIS	Avg. #-W	Max #-W	Min #-W	Avg. BLEU (Pair)
Bar	5,523 (76.21%)	19,407 (75.37%)	3.514	25	62	5	0.323
Pie	520 (7.18%)	1,750 (6.80%)	3.365	15	33	7	0.313
Line	380 (5.24%)	1,562 (6.07%)	4.111	25	49	6	0.380
Scatter	266 (3.67%)	1,041 (4.04%)	3.914	17	39	8	0.438
Stacked Bar	359 (4.95%)	1,172 (4.55%)	3.265	28	45	11	0.405
Grouping Line	72 (1.00%)	271 (1.05%)	3.764	27	51	9	0.365
Grouping Scatter	127 (1.75%)	547 (2.12%)	4.307	19	31	8	0.476
All Types	***7,247***	***25,750***	***3.746***	***22.29***	***44.29***	***7.71***	***0.337***

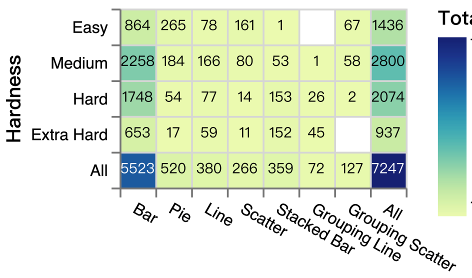


Figure 10: Visualization types *vs.* hardness

nvBench synthesizes 3.746 NL queries per VIS on average. Moreover, the average number of words for each NL query is ~22, and roughly ~16 of them are words for specifying *what* data is needed and the remaining words are for *how* to visualize the data. For some cases, the number of words of an NL query is large; the reason is that the VIS query is complex that contains filtering and join operations. Besides, our NL queries are well-specified, which also leads to a large number of words of the sentence.

<u>*Diversity of* NL *Queries.*</u> For different NL queries *w.r.t.* the same VIS query, we need to ensure that they have enough difference (or diversity). One common way to quantify two texts is through Bilingual Evaluation Understudy (BLEU) score [43], which measures the diversity of pairwise sentences, by counting matching *n*-grams. BLEU's output is always a number between 0 and 1, with values closer to 1 representing more similar texts (*i.e.,* more common *n*-grams) and closer to 0 means good diversity. Note that BLEU mainly quantifies syntactic diversity; that is, it does not quantify the "meaning". This actually fits our problem well because we use NL2VIS to quantify the quality of NL queries.

Table 3 reports the results, the sentences for pie charts have the lowest BLEU score (0.313), which means that the diversity of the NL queries for pie charts is high. The average BLEU score for all cases is 0.337, which shows reasonable diversity for NL queries for the same VIS.

3.3 Expert and Crowd Evaluation

Human Tasks. The key questions we answer with our human evaluation are: (T1) *How close are machine-generated* NL *queries to expected handwritten* NL *queries in* nvBench*?* (T2) *How does* NL *queries match corresponding visualizations in* nvBench*?* and (T3) *Collect handwritten* NL *queries for given visualizations.* Tasks T1 and T2 are to ask humans to verify that whether our synthesized NL2VIS

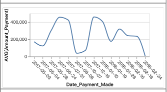
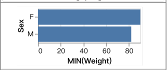
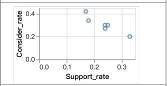
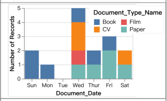
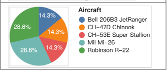
#	Participants	Task 1	Task 2	NL Query	VIS
①	Experts	Agree	Strongly Agree	*Return a trend line to show the average amount of payment by each payment day.*	
	Crowd	Agree	Strongly Agree		
②	Experts	Agree	Strongly Agree	*Show me the minimal weight by sex in a histogram, sort the bars in desc order.*	
	Crowd	Strongly Agree	Agree		
③	Experts	Strongly Agree	Strongly Agree	*Plot a scatter chart, to show the correlation between support and consider rates for each candidate.*	
	Crowd	Strongly Agree	Strongly Agree		
④	Experts	Disagree	Strongly Agree	*How many documents are created in each day?, Bin the document date by weekday and group by document type with a stacked bar chart.*	
	Crowd	Agree	Strongly Agree		
⑤	Experts	Neutral	Agree	*Please list the proportion of the number of each winning aircraft.*	
	Crowd	Agree	Strongly Agree		

Figure 11: Sample questions and answers

benchmark, nvBench, is good. Task T3 aims at enriching nvBench by soliciting handwritten NL queries from experienced users.

<u>*Task T1.*</u> Given a (NL, VIS) pair, we ask the participants the question: "*how close the given* NL *query is to their expectation of handwritten* NL *query*" with five choices {strongly disagree, disagree, neutral, agree, strongly agree}.

<u>*Task T2.*</u> Given a (NL, VIS) pair, we ask the participants: "*how does the* NL *query match the provided* VIS" with five choices {strongly disagree, disagree, neutral, agree, strongly agree}.

We pack one T1 task and one T2 task as a HIT (*i.e.,* a combined question), and show this HIT to the participants, as depicted in Figure 11. All VIS objects are rendered by Vega-Lite. We told the participants that the NL queries can be either machine-generated or handwritten. We also explicitly told the participants that T1 and T2 are *not* correlated, in order to remove the potential bias such that even if one thinks that an NL query is machine-generated in T1, he/she will not lower the matching rate of a (NL, VIS) pair in T2 based on the answer to T1. The rationality of this setting is that, an NL query is not *natural* does not mean that it does not match the corresponding VIS object.

Example 6. *Consider Figure 11 with examples from our real test. In Figure 11-④, an expert considers T1 as a machine-generated* NL *query (rated "disagree") while still considers T2 as a good match (rated "Strongly agree"), which shows that participants are aware that T1 and T2 should not be correlated.*

<u>*Task T3.*</u> Given a VIS object, we ask the participants to provide corresponding handwritten NL queries.

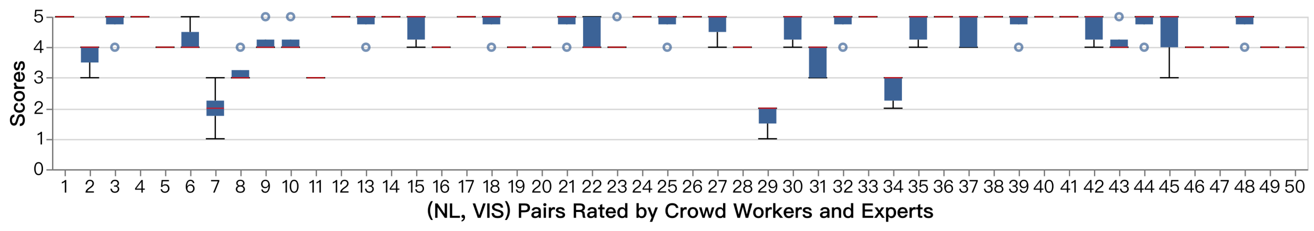


Figure 12: Inter-rater reliability analysis for T2

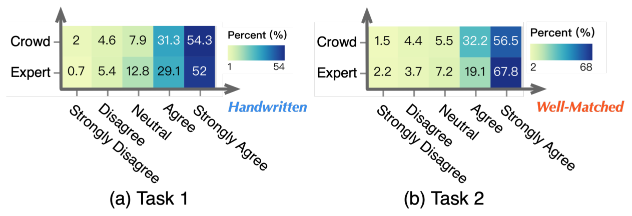


Figure 13: Expert/crowd evaluation

For T1 and T2, we randomly sampled ~10% (NL, VIS) pairs and their associated tables from nvBench. We additionally added n = 100 high-quality handwritten NL queries by two PhDs, such that it has a mixture of machine-generated and handwritten NL queries. Of course, we did not expose which NL queries are machine-generated and which NL queries are handwritten to the participants. T3 was performed along with T1 and T2, if participants can provide new NL queries for given VIS objects, which is not mandatory.

Participants. We invited both experts and crowd workers.

Experts: We invited 23 experts (4 females, 19 males, age 21–42), including 14 Ph.D. students, 4 master students, 1 undergraduate student, 2 research scientists, 1 R&D engineer, and 1 tenured professor. Our participants' research fields include Human-Computer Interaction (1), Natural Language Processing (1), Data Visualization (3), Data Mining (2), and Database (16). All participants have more than 3 years experience on data analysis and visualization. Because the cost for experts is high, we assign one task to one expert and trust his/her quality.

Crowd Workers: We recruited 312 workers from a crowdsourcing platform appen[3], who are good at English and had ≥ 90% HIT approval rating. Each task was completed by 3 to 7 crowd workers (by default 3). The majority voting [8–10] is used to aggregate the answers from tasks T1 and T2. If the answers from three workers cannot be aggregated via majority voting, *e.g.,* each one gives a different answer, we will ask more questions, but are capped at seven. In our testing, most questions are answered by three workers.

Tasks T1 and T2 are performed by both experts and crowd workers. Task T3 is performed only by experts, because T3 is harder and crowd workers failed T3 on our test.

Finally, we collected 2105 answers for T1 and T2 from crowd workers, and 460 results for T1 and T2 from experts. Moreover, we obtained 460 handwritten NL queries in T3, only from experts.

Next, we report our experimental findings.

Exp-T1 *(Machine-generated or handwritten?).* The results from both crowd workers and experts are given in Figure 13(a). It shows that crowd workers/experts consider that 85.6% (31.3%+54.3%) / 81.1% (29.1%+52%) synthesized NL queries are handwritten (*i.e.,* those rated as agree/strongly agree). There are only 6.6% (2%+4.6%), 6.1% (0.7%+5.4%) of the NL queries are not written by humans (*i.e.,* those rated as strongly disagree/disagree), rated by experts and crowd workers respectively. Also, it is difficult for crowd workers and experts to determine whether 7.9%/12.8% NL queries are handwritten or machine-generated (*i.e.,* those rated as neutral).

In summary, both experts and crowd workers believe that most NL queries are handwritten, instead of machine-generated. This is expected, because a large portion of the text is inherited from handwritten NL queries from Spider. Also, we consider that the result from experts is more precise than crowd workers, because experts have good experience in data visualization but crowd workers typically do not.

We also collected some comments from participants for those NL queries that they reported as machine-generated. The most common comment is that some NL queries are long and complex, which mainly correspond to the (extra) hard cases. The high complexities of such NL for VIS queries are mainly carried over from corresponding SQL queries with complex data operations. We keep (extra) hard cases in nvBench for advanced database users. Other comments are as follows: (1) some sentences use words that are not natural, (2) multiple punctuation marks appear at the same time, (3) some sentences are imperative mood, and (4) the words contain special marks such as underlines.

Exp-T2 *(Does* NL *queries match* VIS*?).* Figure 13(b) shows the result. It tells us that experts consider 86.9% (67.8%+19.1%) NL queries match (*i.e.,* those rated as strongly agree/agree) the corresponding VIS well, and crowd workers think 88.7% (56.5%+32.2%) NL queries are well-matched. Moreover, these experts also point out that the easy and medium cases are very commonly used when they specify visualizations, but rarely specify (extra) hard cases, for which they tend to first use SQL queries to get data and then visualize.

In summary, both experts and crowd workers agree that most NL queries well match (rated as strongly agree/agree) VIS queries. We also proofread those (NL, VIS) pairs rated as neutral/disagree/strongly/disagree. We find that most of the NL queries contain some descriptions about *Filter/Join* operations. Because it is difficult for the participants to directly observe the NL descriptions about *Filter/Join* operations either from the VIS results and the associated tables, these types of (NL, VIS) pairs are often falsely rated as neutral/disagree/strongly disagree.

Inter-rater Reliability. Next we analyze the degree of agreement between participants for task T2. We randomly sampled 50 overlapping (NL, VIS) pairs in T2 that are rated by both crowd workers and

[3]https://appen.com/, Figure Eight—formerly Crowdflower—is now Appen

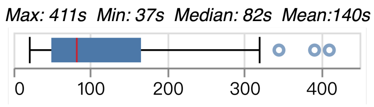


Figure 14: Task T3 – user time (seconds)

experts. We map the answers strongly disagree/disagree/neutral/agree/strongly agree to score 1/2/3/4/5 for easy plotting. Figure 12 uses a boxplot to visualize the distribution of answers. Each point in the x-axis is for a sampled (NL, VIS) pair, the y-axis for the ratings from both experts and crowd workers, and the boxplot for agreement/disagreement. The red line represents the median value of answers while the box boundaries correspond to the 25th and 75th percentiles. The blue dots (*e.g.,* in $x \in \{3, 8, 9, 10, \ldots\}$ are identified as outliers and are automatically annotated by Vega-Lite while rendering the figure). As shown in Figure 12, we can see that most of the tasks have a high degree of agreement between participants with the following three cases: *1. Fully Agree*: all crowd workers and the expert give the same ratings in 22 (NL, VIS) pairs (*e.g.,* $x = \{1, 4, 5, 11, \ldots\}$). *2. Mainly Agree*: the difference of ratings between all crowd workers and the expert is maximum 1 such as agree and strongly agree (*e.g.,* $x = \{2, 3, 6, 8, 9, \ldots\}$). *3. Slightly Disagree*: only two cases that have a difference of ratings at 2 (*i.e.,* $x = \{7, 45\}$).

Exp-T3. We collect 460 new NL queries written by experts and also record the time. The boxplot in Figure 14 tells us that the longest time to complete T3 is 411 seconds (close to 7 minutes), the shortest time is 37 seconds, the median is 82 seconds, and the average time is 140 seconds (more than 2.3 minutes). By estimation, the total time for humans to build NVBENCH from scratch is $140 \div 60 \times 25750 = 60083$ minutes, which is about 1001 hours, around 42 days.

Besides collecting 460 handwritten NL queries, it also shows that our NL2SQL-to-NL2VIS synthesizer can significantly reduce the man-hour to ~2.4 days (for editing NL queries *w.r.t.* tree deletions, see more details at the end of Section 3.1) from ~42 days that requires experts to provide all NL queries (as analyzed above). Note that, the 42 days man-hours is based on the case that we have generated all meaningful VIS queries; it will take much longer time if we also need humans to provide meaningful VIS queries.

4 NL2VIS NEURAL TRANSLATION

The key questions we answer with these experiments are: (1) Can we translate NL queries to VIS queries by learning from NVBENCH (Section 4.3)? (2) Whether the learning-based NL2VIS approach can outperform the state-of-the-art rule-based or semantic parser based methods for NL2VIS (Section 4.4)? (3) What is the effect of low-rated NL or VIS queries through the generation process (Section 4.5)? (4) We further conduct a case study using COVID-19 data (Section 4.6).

4.1 SEQ2VIS: NL2VIS Neural Translation

NL2VIS is very similar to NL2SQL; the SEQ2SEQ model [53] is a natural choice, where the input NL is a sequence and the output VIS query is also a sequence. We call it SEQ2VIS for convenience, referring to the NL2VIS problem.

Figure 15 overviews SEQ2VIS, which follows an encoder-decoder architecture with attention mechanism [57] and has been used

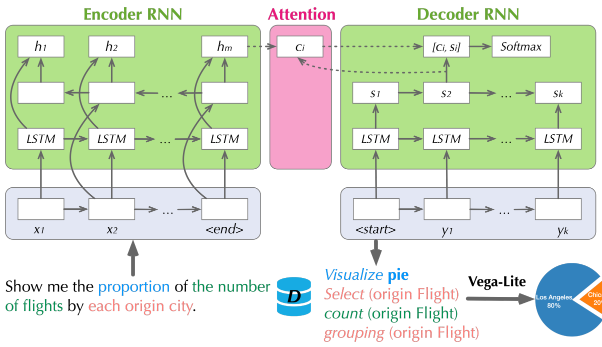


Figure 15: The SEQ2VIS Model

in NL2SQL tasks [4, 7, 53]. It also shows a running example on translating an NL query to a VIS query.

Next, we will briefly explain SEQ2VIS, for more details please refer to [7].

The input NL query is represented as a sequence of tokens (or words) $[q_1, q_2, ..., q_l] \in \mathcal{V}^{in}$. The output VIS query is also a sequence of tokens $[y_1, y_2, ..., y_k] \in \mathcal{V}^{out}$. Here, $\mathcal{V}^{in}$ (resp. $\mathcal{V}^{out}$) is the input (resp. output) vocabulary. For example, in Figure 15, the input sequence is [*"I", "prefer", "a", "pie",* ...] and the output sequence is [*"Visualize", "pie", "Select",* ...].

Encoder RNN. Given an input sequence $n_V = [q_1, q_2, \ldots, q_l]$, we concatenate n_V with the database table schema information $A = [a_1, a_2, ..., a_m]$. We use $X = [x_1, x_2, ..., x_n] = [q_1, q_2, ..., q_l, a_1, a_2, ..., a_m]$ to represent the input sequence, where $n = l + m$. We then use a pre-trained global word embedding (GloVe) [44] to map each token x_i to its vector representation. Once the token embedding is ready, we feed embedded tokens into a bi-directional recurrent neural network (RNN) using Long Short-Term Memory (LSTM) cells. The output is a sequence of encoding vectors (hidden states) $H = [h_1, h_2, ..., h_n]$.

Decoder RNN. The decoder is also an RNN based on LSTM cells with attention mechanism [4] that generates the VIS query $[y_1, y_2, ..., y_k]$ based on the hidden states H. At each time step t, it predicts VIS query token y_t based on the recurrent state in the decoder RNN s_i, the previous tokens, and an attention vector c_i.

Note that one *hardness* of NL2VIS is how to map "student id" in an NL query to a column like "sid". Fortunately, this has been manually corrected (or normalized) by Spider [68] (*i.e.,* change "sid" to "student id") to be semantically meaningful. Therefore, piggybacking Spider can ensure that such cases don't appear in NVBENCH. In other words, if some attribute is "foo_bar", then we cannot to map an NL query to it; solving this problem is not the focus of this work.

4.2 Evaluation Settings

Methods. We consider the following methods.

– SEQ2VIS: We use a standard encoder-decoder architecture with attention mechanism [57] and has been widely used in NL2SQL tasks [4, 7, 53].

– NL4DV: NL4DV [42] is a toolkit that supports to generate data visualization using NL queries, mainly based on NLP semantic parses. Note that, NL4DV [42] cannot handle *Join* and *Nested* queries.

– DeepEye: DeepEye [36] is a rule-based methodology for creating VIS charts. Similar to NL4DV [42], it can not successfully process *Join*, *Nested*, and *Filter* queries.

Remark. For the deep learning based approach, we use a standard and powerful one, instead of those more recent ones such as RAT-SQL [60] and IR-Net [21], for two reasons: (1) SEQ2VIS is more extensible, because RAT-SQL and IR-Net are heavily optimized for NL2SQL on Spider benchmarks; and (2) our main purpose is to give a proof-of-concept that deep learning based approaches work for NL2VIS (Section 4.3), and work better than non-learning based state-of-the-art approaches such as NL4DV and DeepEye (Section 4.4).

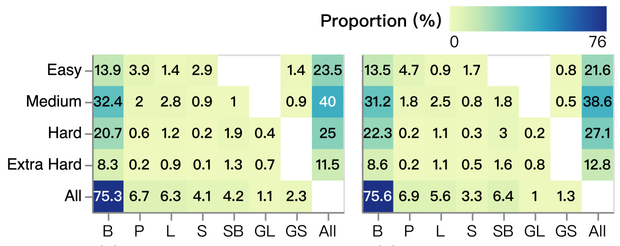
Evaluation Metrics. As discussed in [26] for NL2SQL, measuring the translation quality is not easy. Similar to [26], we also use different levels of accuracy measures. From the VIS AST query perspective, we define "VIS tree matching accuracy" to measure whether the SEQ2VIS model can precisely predict the VIS AST query based on the NL input. From the VIS result perspective, we want to know whether the SEQ2VIS model can predict a VIS result that exactly equivalent to the gold result even if the predicted VIS AST is inaccurate. Furthermore, we also define "VIS component matching accuracy" to measure if the SEQ2VIS model can precisely output each component (x/y-axis) of the VIS.

Tree matching accuracy. It measures if the predicted VIS AST tree exactly matches the ground truth VIS AST tree. We define $Acc_{tree} = N_{tree}/N$, where N_{tree} is the number of generated VIS AST trees that are exactly equivalent to the ground truth VIS AST tree, and N is the total number of trees.

Result matching accuracy. In some cases, the NL2VIS system may predict some VIS queries with "novel" syntax structures, which leads to the predicted VIS query and the ground truth VIS query may not be the same but the visualization results of the predicted VIS query and the ground truth VIS query may be the same. To alleviate this issue, we use visualization VIS result matching accuracy to measure whether the predicted visualization VIS is the same as the ground truth visualization VIS. This metric is computed as $Acc_{res} = N_{res}/N$.

VIS *component matching accuracy.* A visualization (VIS object) consists of three components: VIS *types*, $x/y/z$-*axis*, and *data with transformation from a database D*. In some cases, the VIS system can precisely predict the $x/y/z$-*axis* and *data* parts but fail to predict the VIS *types*. Motivated by this, we propose VIS *component matching* metric to reveal the detailed performance of NL2VIS model on each VIS component. More concretely, for VIS *types* component, we measure the *Visualize* part of the VIS query (Figure 5); for the $x/y/z$-*axis* part, we measure the *Select* component of the VIS query; and for the *data* part, we measure the *Group*, *Filter*, *Order*, and *Superlative* components. We compute VIS component matching accuracy: $Acc_{comp} = N_{comp}/N$, where N_{comp} is the number of components that match to the gold result N. For example, assume it has 100 bar chart ($N = 100$) and it only predicts 88 bar chart ($N_{bar} = 88$), so the $Acc_{bar} = N_{bar}/N = 88\%$.

Datasets. We randomly split the (NL, VIS) pairs of the nvBench into training set with 20598 (80%) pairs, validation set with 1162 (4.5%) pairs, and test set containing 3990 (15.5%) pairs. The heatmaps in



(a) 20,598 Training Examples (b) 3,990 Testing Examples

B (Bar Chart), P (Pie Chart), L (Line Chart), S (Scatter Chart)
SB (Stacked Bar), GL (Grouping Line), GS (Grouping Scatter)

Figure 16: The distribution of training and test set

Figure 16 show the distribution of the training and testing (NL, VIS) pairs. It depicts that the training set and test set have similar distributions on visualization types and hardness. Besides, by comparing with Figure 10, we can see that the training set and test set are close to the entire nvBench distribution.

Currently, SEQ2VIS does not consider to predict the V (value) of the VIS query. Instead of generating V by a learning-based model, we use a heuristic method to extract values from NL queries and fill them into the right part of the predicted VIS tree, *i.e.,* right slots in the predicted visualization query. This heuristic method can achieve ~92.3% accuracy on average.

Training Settings. We use a GloVe embedding model on the concatenation of the VIS query and response output of the training data for the embedding layer of our SEQ2VIS model. We use an embedding dimension of 100, hidden dimension of 150, a bi-directional LSTM encoder and uni-directional LSTM decoder with attention. We use a batch size of 16, clip the norm of the gradient at 2.0, and do early stopping on the validation loss with a patience of 5.

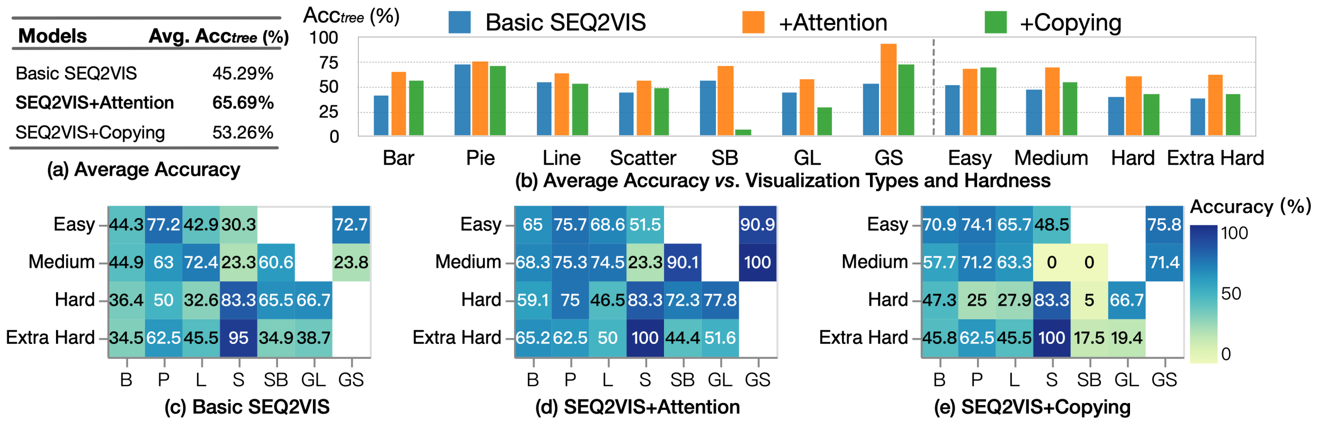
4.3 Performance of SEQ2VIS

Figure 17 summarizes the performance of SEQ2VIS, including basic SEQ2VIS, SEQ2VIS with attention [4], and SEQ2VIS with the copying mechanism [25], on the test set using VIS tree matching metrics.

Figure 17(a) shows the VIS tree matching accuracy, which shows that SEQ2VIS with attention performs the best and achieves averagely 65.69% VIS tree matching accuracy, which matches the state-of-the-art performance of NL2SQL tasks.

Figure 17(b) depicts the accuracy of three versions of SEQ2VIS on different visualization types and difficulty levels. It also agrees that SEQ2VIS with attention works the best. More concretely, the pie chart has the highest accuracy among the three methods on average. By varying the visualization hardness, we observe that the easy cases have the highest scores on the SEQ2VIS with copying, and get the lowest scores on the basic SEQ2VIS. Overall, the average performance of the model is related to the difficulty of the VIS query, and the performance is better in easier cases.

Figures 17(c)-(e) show the accuracy of three versions of SEQ2VIS under different visualization types and difficult levels. SEQ2VIS with attention outperforms the other two under almost all combinations of visualization types and difficulty levels. Although the average accuracy of SEQ2VIS with copying is 7.97% higher than the basic SEQ2VIS, SEQ2VIS with copying fails in predicting the medium stacked bar/scatter charts (Figure 17(e)). If we take the distribution of training and test set as consideration (Figure 16), some accuracy



Models	Avg. Acctree (%)
Basic SEQ2VIS	45.29%
SEQ2VIS+Attention	**65.69%**
SEQ2VIS+Copying	53.26%

(a) Average Accuracy

Figure 17: VIS Tree matching accuracy, by varying hardness *vs.* visualization types on different methods (test set)

Table 4: Average VIS component matching accuracy (%) on NVBENCH

	VIS								Axis	Data				
	Bar	Pie	Line	Scatter	SB	GL	GS	***All***	Select	Where	Join	Grouping	Binning	Order
SEQ2VIS	97.6	91.0	88.6	91.8	85.0	**92.5**	89.8	95.5	71.3	85.0	81.9	76.4	91.7	73.0
+Attention	98.8	89.9	**90.9**	88.2	**91.9**	**92.5**	**96.9**	**97.0**	**79.5**	**91.6**	**91.4**	**86.0**	**95.1**	84.7
+Copying	**99.1**	**92.8**	89.5	**96.5**	19.8	**92.5**	85.7	92.8	78.6	83.9	84.9	80.4	92.3	**85.0**
Avg.	98.5	91.2	86.7	92.2	65.6	92.5	90.8	95.1	76.5	86.8	86.1	80.9	93.0	80.9

Table 5: Comparison with the state-of-the-art solutions

	DeepEye				NL4DV	SEQ2VIS
	Top-1	Top-3	Top-6	All Results	Top-1	Top-1
Easy	9.5%	15.3%	23.0%	44.2%	11.5%	67.4%
Medium	15.4%	21.9%	24.7%	28.8%	22.5%	69.6%
Hard	1.4%	1.6%	1.8%	2.0%	7.6%	60.5%
Extra Hard	6.1%	6.8%	7.6%	8.0%	4.1%	61.8%
Overall	9.1%	13.1%	15.9%	22.2%	13.7%	65.7%

results, especially some hard/extra hard cases, may be considered as outliers because it only accounts for a small proportion. For example, all three methods work nearly perfectly on the extra hard scatter charts (see (S, Extra Hard) in Figure 17(c)-(e)). The reason is probably because this type of visualizations only accounts for 0.1% (0.1% × 20598 =∼20) and 0.5% (0.5% × 3990 =∼20) the training and test sets, respectively.

Table 4 showcases the average VIS component matching accuracy. Overall, the SEQ2VIS with the attention mechanism performs well in each VIS component prediction task. We compute the average performance (denoted as **Avg.**) of three models on each VIS component. It shows that the three models have the best accuracy in predicting the Bar, that is, they can easily predict the visualization type–bar chart. For predicting Axis, *i.e.*, $x/y/z$-axis, three versions of SEQ2VIS perform poorly, with an average accuracy of 76.5%. The main reason is that when predicting the y-axis, the corresponding aggregate functions (*e.g.*, avg, sum) must be precisely predicted. For data part, three versions of SEQ2VIS perform best on predicting Binning operations and work worst on predicting Order operations. Besides, the VIS result matching metric has a slight improvement in each version of the SEQ2VIS model over the VIS tree matching metric, and we omit the discussion due to the space constraint.

4.4 Comparison with the State of the art

We use the same test set containing 3990 (NL, VIS) pairs to compare the performance between SEQ2VIS with attention, DeepEye [36], and NL4DV [42]. Because DeepEye can return top-k visualizations for the input natural language (keywords) query, we report the top-1, top-3, top-6, and all results (averagely top-19) given by DeepEye. Table 5 reports the evaluation results. Our SEQ2VIS with attention significantly outperforms DeepEye and NL4DV, because it learns from the NVBENCH instead of relying on the rule-based approaches. More concretely, our SEQ2VIS with attention can better handle hard and extra hard NL questions, but the other two work poorly on NL questions of such hardness. Interestingly, although the accuracy of DeepEye's top-1 result is lower than that of NL4DV's top-1 result, the accuracy of DeepEye's top-6 results is better than NL4DV in the easy and medium NL questions. In summary, the learning-based method–SEQ2VIS significantly outperforms the state-of-the-art rule-based and semantic parser-based approaches for NL2VIS tasks. In particular, the learning-based method has the stronger ability to handle those hard and extra hard queries with *Join*, *Filtering*, and *Nested* operations, while the state-of-the-art solutions work very poorly on such cases.

4.5 Effect of Low-rated (NL, VIS) Pairs

As reported in Section 3.3, the crowd workers and experts have identified 231 imperfect (NL, VIS) pairs (*i.e.*, those rated as strongly disagree/disagree either in the Task 1 or Task 2) in the user study. We now conduct an experiment to test the effect of low-rated (NL, VIS) pairs. We first removed all low-rated (NL, VIS) pairs, identified by the crowd workers and experts, in the training set, to train three versions of SEQ2VIS models as the baseline. We then randomly

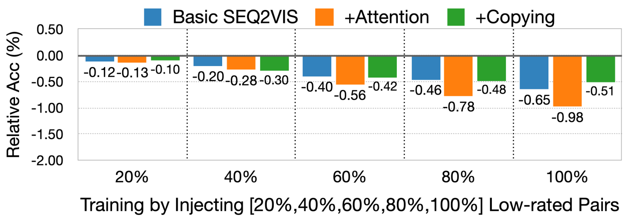


Figure 18: Relative accuracy *w.r.t.* low-rated pairs

injected $x\%$ ($x = [20, 40, 60, 80, 100]$) of low-rated (NL, VIS) pairs into the training set for training. We repeated the training three times to compute the average accuracy. We then tested the accuracy difference between the model trained without low-rated (NL, VIS) pairs with the models trained with $x\%$ low-rated (NL, VIS) pairs.

Figure 18 shows the results, it depicts that the low-rated (NL, VIS) pairs have a slight influence on the performance of three versions of SEQ2VIS models. The SEQ2VIS with an attention mechanism is more influenced by the low-rated (NL, VIS) pairs. Overall, the impact of low-rated (NL, VIS) pairs on the performance of three versions of SEQ2VIS models are small, showing the robustness of SEQ2VIS.

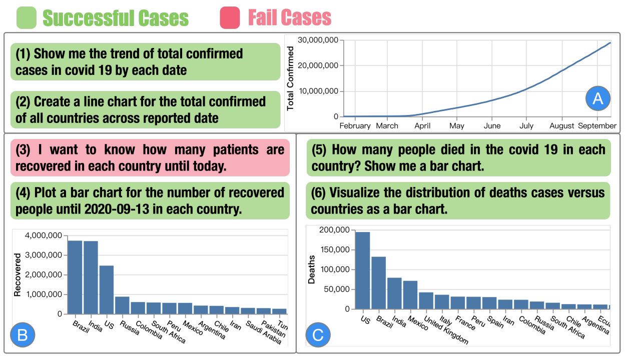
4.6 Case Study using COVID-19 Data

We use the COVID-19 dataset, with the schema of (Date, Country, Confirmed, Active Cases, Recovered, Deaths, Daily Cases), to test whether the SEQ2VIS can support some popular use cases from the well-designed COVID-19 dashboards [14, 33, 37]. We invite 3 experts in the task (T3) to participate in this case study. We first ask experts to observe the 5 designed visualizations provided by `JHU COVID-19 Dashboard` [14] (https://coronavirus.jhu.edu/map.html). We then ask each expert to write 2 NL queries (6 NL queries in total) for querying the visualization. The case study results are given in Figure 19. We have 5 NL queries (colored in green) are successfully predicted by SEQ2VIS, and 1 NL query (colored in red) is failed. Next, we analyze the 1 failed NL query. For the failed NL query, as shown in Figure 19-Ⓑ-(3), it fails because it contains "until today", the unknown word in the dataset. It is hard for the SEQ2VIS to convert "until today" to the date "2020-09-13", and thus fails to construct the "Filter" subtree for the VIS AST tree. Overall, the case study demonstrates the effectiveness of the SEQ2VIS on the well-designed visualization tasks provided by the `JHU COVID-19 Dashboard`.

5 RELATED WORK

VIS Benchmarks. VizNet [23] provides a large-scale visualization learning and benchmarking repository, which can help train automated visualization tools or evaluate the effectiveness of VIS designs. IDEBench [17] is used to evaluate the performance of DBMSs for interactive data exploration (IDE) workloads. [6] is similar to IDEBench, which is for validating the support of DBMSs for IDE workloads, but it focuses particularly on crossfilter-style applications. None of these benchmarks is designed for NL2VIS.

NL2SQL **Benchmarks.** Both the database community [28, 66] and NLP community [15, 21, 67] have extensively studied the problem of NL2SQL, mainly benefit from the availability of a rich collection of NL2SQL benchmarks: ATIS [45], Yelp and IMDB [66], Restaurants [54], Scholar [24], WikiSQL [69], and Spider [68].



(1) Show me the trend of total confirmed cases in covid 19 by each date

(2) Create a line chart for the total confirmed of all countries across reported date

(3) I want to know how many patients are recovered in each country until today.

(4) Plot a bar chart for the number of recovered people until 2020-09-13 in each country.

(5) How many people died in the covid 19 in each country? Show me a bar chart.

(6) Visualize the distribution of deaths cases versus countries as a bar chart.

Figure 19: Case Study with COVID-19 Data

NL2VIS **Techniques.** Text-to-viz [11] uses rule-based methods to translate text statements to infographics instead of visualizations. There has been a surge of works on developing an NL interface for data VIS [1–3, 19, 34, 36, 42, 46, 49, 50, 52]. Unfortunately, there are no publications that are associated with the NL2VIS support from Microsoft Power BI [2], ThoughtSpot [3], and Tableau's Ask Data [1]. NL4DV [42] is a toolkit that supports to generate data visualization using NL queries, mainly based on NLP semantic parses [20, 29, 61], similar to DataTone [19]. DeepEye [36, 46] demonstrates a simple rule-based method for generating VIS charts from keyword queries.

Training Data Generation. The work [13] increases the speed of annotating (NL, SQL) pairs over structured data. They sample Operation Trees and ask annotators to write NL queries for these sampled operation trees. We differ in two main aspects: (1) their Operation Tree generation and selection method cannot be used to select visualizations, and (2) they cannot produce and edit the NL queries automatically. Our NL2SQL-to-NL2VIS, proposed in Section 2, is proposed to mainly solve the above two points.

DBPal [63] augments training data based on a set of pre-defined (NL, SQL) templates. On the one hand, designing templates with wide coverage, high diversity and high quality requires a lot of effort from experts, as discussed in their paper. On the other hand, DBPal's main goal is to do data augmentation, which cannot be directly used to generate NL2VIS benchmarks from NL2SQL benchmarks, as studied in this work.

6 CONCLUSION

We have proposed a novel NL2SQL-to-NL2VIS synthesizer that can synthesize NL2VIS benchmarks from existing NL2SQL benchmarks. We have evaluated the effectiveness of our synthesized NL2VIS benchmark (*i.e.,* NVBENCH) which has high accuracy (*i.e.,* the synthesized NL queries are very close to human-provided ones), good coverage (*i.e.,* many domains, and different complexities of databases), and good diversity (*i.e.,* many different VIS types). In addition, in order to learn the translation from NL queries to VIS queries, we are the first to apply NL2VIS neural translation model, SEQ2VIS. Experimental results show that SEQ2VIS works well and significantly outperforms the *state-of-the-art* approaches of the NL2VIS task.

Acknowledgement. This work is supported by NSF of China (61925205, 61632016), Huawei, BNRist, and TAL Education. Chengliang Chai is supported by the Zhejiang Lab's International Talent Fund for Young Professionals.

REFERENCES

[1] Ask Data. https://www.tableau.com/products/new-features/ask-data.

[2] Microsoft Power BI Q&A. https://docs.microsoft.com/en-us/power-bi/create-reports/power-bi-tutorial-q-and-a.

[3] SpotIQ AI-Driven Insights (2nd Edition). https://www.thoughtspot.com/resources#white_paper.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[5] L. Battle, P. Duan, and et al. Beagle: Automated extraction and interpretation of visualizations from the web. In *CHI*, page 594, 2018.

[6] L. Battle, P. Eichmann, and et al. Database benchmarking for supporting real-time interactive querying of large data. In *SIGMOD*, 2020.

[7] D. Britz, A. Goldie, M. Luong, and Q. V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

[8] C. Chai, L. Cao, G. Li, J. Li, Y. Luo, and S. Madden. Human-in-the-loop outlier detection. In *SIGMOD*, pages 19–33. ACM, 2020.

[9] C. Chai, G. Li, J. Fan, and Y. Luo. Crowdchart: Crowdsourced data extraction from visualization charts. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[10] C. Chai, G. Li, J. Fan, and Y. Luo. Crowdsourcing-based data extraction from visualization charts. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1814–1817. IEEE, 2020.

[11] W. Cui, X. Zhang, Y. Wang, and et al. Text-to-viz: Automatic generation of infographics from proportion-related natural language statements. *IEEE Trans. Vis. Comput. Graph.*, 26(1):906–916, 2020.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[13] J. Deriu, K. Mlynchyk, P. Schläpfer, Á. Rodrigo, and et al. A methodology for creating question answering corpora using inverse data annotation. In *ACL*, pages 897–911, 2020.

[14] E. Dong, H. Du, and L. Gardner. An interactive web-based dashboard to track covid-19 in real time. *The Lancet infectious diseases*, 20(5):533–534, 2020.

[15] L. Dong and M. Lapata. Coarse-to-fine decoding for neural semantic parsing. In *ACL*, pages 731–742, 2018.

[16] S. Edunov, M. Ott, M. Auli, and D. Grangier. Understanding back-translation at scale. In *EMNLP*, pages 489–500. ACL, 2018.

[17] P. Eichmann, E. Zgraggen, C. Binnig, and T. Kraska. Idebench: A benchmark for interactive data exploration. In *SIGMOD*, 2020.

[18] M. Fadaee, A. Bisazza, and C. Monz. Data augmentation for low-resource neural machine translation. In *ACL*, pages 567–573. Association for Computational Linguistics, 2017.

[19] T. Gao, M. Dontcheva, and et al. Datatone: Managing ambiguity in natural language interfaces for data visualization. In *UIST*, 2015.

[20] Z. Gharibshah, X. Zhu, A. Hainline, and M. Conway. Deep learning for user interest and response prediction in online display advertising. *Data Science and Engineering*, 5(1):12–26, 2020.

[21] J. Guo, Z. Zhan, Y. Gao, , and et al. Towards complex text-to-sql in cross-domain database with intermediate representation. In *ACL*, 2019.

[22] P. Hanrahan. Vizql: a language for query, analysis and visualization. In *SIGMOD*, page 721. ACM, 2006.

[23] K. Z. Hu, S. N. S. Gaikwad, and et al. Viznet: Towards A large-scale visualization learning and benchmarking repository. In *CHI*, 2019.

[24] S. Iyer, I. Konstas, A. Cheung, and et al. Learning a neural semantic parser from user feedback. In *ACL*, pages 963–973, 2017.

[25] R. Jia and P. Liang. Data recombination for neural semantic parsing. ACL, 2016.

[26] H. Kim, B. So, W. Han, and H. Lee. Natural language to SQL: where are we today? *Proc. VLDB Endow.*, 13(10):1737–1750, 2020.

[27] D. Li, H. Mei, and et al. Echarts: A declarative framework for rapid construction of web-based visualization. *Vis. Informatics*, 2018.

[28] F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84, 2014.

[29] M. Li, H. Wang, and J. Li. Mining conditional functional dependency rules on big data. *Big Data Mining and Analytics*, 03(01):68, 2020.

[30] Z. C. Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[31] Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li. Interactive cleaning for progressive visualization through composite questions. In *36th IEEE International Conference on Data Engineering, ICDE*, pages 733–744. IEEE, 2020.

[32] Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li. Visclean: Interactive cleaning for progressive visualization. *Proc. VLDB Endow.*, 13(12):2821–2824, 2020.

[33] Y. Luo, W. Li, T. Zhao, X. Yu, L. Zhang, G. Li, and N. Tang. Deeptrack: Monitoring and exploring spatio-temporal data - A case of tracking COVID-19 -. *Proc. VLDB Endow.*, 13(12):2841–2844, 2020.

[34] Y. Luo, X. Qin, C. Chai, N. Tang, G. Li, and W. Li. Steerable self-driving data visualization. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[35] Y. Luo, X. Qin, N. Tang, and G. Li. Deepeye: Towards automatic data visualization. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 101–112, 2018.

[36] Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang. Deepeye: Creating good data visualizations by keyword search. In *SIGMOD*, 2018.

[37] Y. Luo, N. Tang, G. Li, W. Li, T. Zhao, and X. Yu. Deepeye: A data science system for monitoring and exploring COVID-19 data. *IEEE Data Eng. Bull.*, 2020.

[38] J. D. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Trans. Vis. Comput. Graph.*, 2007.

[39] A. Marcus and A. G. Parameswaran. Crowdsourced data management: Industry and academic perspectives. *Found. Trends Databases*, 2015.

[40] T. Mikolov, I. Sutskever, K. Chen, and et al. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[41] D. Moritz, C. Wang, G. L. Nelson, and et al. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE Trans. Vis. Comput. Graph.*, 25(1):438–448.

[42] A. Narechania, A. Srinivasan, and J. T. Stasko. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Trans. Vis. Comput. Graph.*, 27(2):369–379, 2021.

[43] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, 2002.

[44] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[45] P. J. Price. Evaluation of spoken language systems: the ATIS domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, USA*. Morgan Kaufmann, 1990.

[46] X. Qin, Y. Luo, N. Tang, and G. Li. Deepeye: Visualizing your data by keyword search. In *EDBT*, pages 441–444, 2018.

[47] X. Qin, Y. Luo, N. Tang, and G. Li. Making data visualization more efficient and effective: a survey. *VLDB J.*, 29(1):93–117, 2020.

[48] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE TVCG*, 23(1):341–350, 2017.

[49] V. Setlur, S. E. Battersby, and et al. Eviza: A natural language interface for visual analysis. In *UIST*, 2016.

[50] V. Setlur, M. Tory, and A. Djalali. Inferencing underspecified natural language utterances in visual analysis. In *IUI*, pages 40–51. ACM, 2019.

[51] T. Siddiqui, A. Kim, and et al. Effortless data exploration with zenvisage: An expressive and interactive visual analytics system. *PVLDB*, 2016.

[52] A. Srinivasan and J. T. Stasko. Natural language interfaces for data analysis with visualization: Considering what has and could be asked. In *EuroVis*, pages 55–59. Eurographics Association, 2017.

[53] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[54] L. R. Tang and R. J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *EMCL*, 2001.

[55] N. Tang, E. Wu, and G. Li. Towards democratizing relational data visualization. In *SIGMOD*, pages 2025–2030. ACM, 2019.

[56] M. Vartak, S. Rahman, and et al. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 2015.

[57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

[58] A. Wang, Y. Pruksachatkun, and et al. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NIPS*, 2019.

[59] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

[60] 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 *ACL*, 2020.

[61] Y. Wang, Y. Yao, H. Tong, F. Xu, and J. Lu. A brief review of network embedding. *Big Data Mining and Analytics*, 2(1):35, 2019.

[62] Y. Wang, Y. Yuan, Y. Ma, and G. Wang. Time-dependent graphs: Definitions, applications, and algorithms. *Data Science and Engineering*, 4(4):352–366, 2019.

[63] N. Weir, P. Utama, A. Galakatos, A. Crotty, A. Ilkhechi, S. Ramaswamy, R. Bhushan, N. Geisler, B. Hättasch, S. Eger, U. Çetintemel, and C. Binnig. Dbpal: A fully pluggable NL2SQL training pipeline. In *SIGMOD*, pages 2347–2361, 2020.

[64] H. Wickham. *ggplot2 - Elegant Graphics for Data Analysis*. Use R. Springer, 2009.

[65] K. Wongsuphasawat and et al. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE TVCG*, 2016.

[66] N. Yaghmazadeh, Y. Wang, and et al. Sqlizer: query synthesis from natural language. *Proc. ACM Program. Lang.*, pages 63:1–63:26, 2017.

[67] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. R. Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *NAACL-HLT*, 2018.

[68] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. 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*, pages 3911–3921, 2018.

[69] V. Zhong and et al. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

[70] X. Zhou, C. Chai, G. Li, and J. SUN. Database meets artificial intelligence: A survey. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2020.