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

Yuyu Luo
Tsinghua University, China
luoyu18@mails.tsinghua.edu.cn

Nan Tang
QCRi, HBKU, Qatar
ntang@bbku.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, VisiQL, 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 AST 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD ’21, June 20–25, 2021, Virtual Event, China © 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8343-1/21/06...$15.00
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], VisiQL [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.
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, SQ) pair. (Section 2)
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, 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 \((\eta_Q, Q)\) 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 \(t_1\)).
**Example 2.** [Input.] Figure 4 gives a real example from a Flight dataset, where the original (nl, sq) 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 \(n_1\) and \(n_2\); and for \(t_2\), it also synthesizes two NL queries \(n_3\) and \(n_4\). Hence, it will output four pairs \(\{n_1, t_1\}, \{n_2, t_1\}, \{n_3, t_2\}, \{n_4, t_2\}\).

**Step 1. VIS Synthesis.** It does tree edits to obtain multiple VIS trees from one SQL tree, considered as 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 \(\Delta_t\) the tree edits on SQL tree \(t_Q\) to get VIS tree \(t_t\).

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 \(t_t\), and the tree operations \(\Delta_t\) that convert \(t_Q\) into \(t_t\), it will revise \(n_Q\) to reflect the change of \(\Delta_t\), 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.
2.3 Tree Edits: Generating VIS Candidates

Next, we discuss how to generate (candidate) VIS trees from one SQ tree. Intuitively, given one SQ tree, we can "delete" any tree nodes or "insert" any tree nodes, as long as we can produce a valid VIS tree rooted with "VVisualize 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 SQL queries will be similar – providing many similar, or redundant (SQL, 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 SQ tree $t_Q$, we first perform different deletions on $t_Q$ to get a set I of intermediate SQ trees {$t_1, \ldots, t_n$}. For each intermediate SQ tree $t_i \in I$, we then make insertions to get a set of VIS trees $TV = \{t_1, \ldots, t_n\}$. The tree edit from $t_Q$ to $t_i$ ($i \in [1, n]$) is denoted by $\Delta_i$, consisting of deletions ($\Delta^-_i$) and insertions ($\Delta^+_i$).

[Deletions ($\Delta^-$).] An SQ 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 SQ 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 SQ tree in $I$.

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

[Insertions ($\Delta^+$).] Next we discuss how to do insertions for each $t^I \in I$ and output a set $TV$ 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, $\#bins = 10$).

(VVisualize) 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.

(Orders) 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 $TV$.

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

2.4 Filtering Bad Visualizations

We use a pre-trained ML model M($\cdot$), DeepEye [35]\(^1\), to prune candidate VIS queries that are bad. Given a VIS query $v$, 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 $TV$ and get a set of good VIS set $TV'$.

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

Table 1: Rules for syntactically correct charts

<table>
<thead>
<tr>
<th>One Variable</th>
<th>Two Variables</th>
<th>Three Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (Categorical), T (Temporal), Q (Quantitative)</td>
<td>C + Q</td>
<td>T + Q</td>
</tr>
<tr>
<td>C grouping + count → [bar, pie]</td>
<td>grouping + binning + count → [bar, pie]</td>
<td>grouping + binning + count → [bar, pie, line]</td>
</tr>
<tr>
<td>T grouping(binning) + count → [bar, pie]</td>
<td>grouping(binning) + count → [bar, pie, line]</td>
<td></td>
</tr>
<tr>
<td>I grouping(binning) + count → [bar, pie, line]</td>
<td>scatter</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C + Q</th>
<th>T + Q</th>
<th>I + Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>grouping + binning + agg → [grouping, line, stacked bar]</td>
<td>grouping + binning + agg → [grouping, line, stacked bar]</td>
<td>grouping + binning + agg → [grouping, scatter]</td>
</tr>
</tbody>
</table>

Figure 6: Sample SQL and VIS trees

Table 1: Rules for syntactically correct charts

<table>
<thead>
<tr>
<th>One Variable</th>
<th>Two Variables</th>
<th>Three Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>C (Categorical), T (Temporal), Q (Quantitative)</td>
<td>C + Q</td>
<td>T + Q</td>
</tr>
<tr>
<td>C grouping + count → [bar, pie]</td>
<td>grouping + binning + count → [bar, pie]</td>
<td>grouping + binning + count → [bar, pie, line]</td>
</tr>
<tr>
<td>T grouping(binning) + count → [bar, pie]</td>
<td>grouping(binning) + count → [bar, pie, line]</td>
<td></td>
</tr>
<tr>
<td>I grouping(binning) + count → [bar, pie, line]</td>
<td>scatter</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C + Q</th>
<th>T + Q</th>
<th>I + Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>grouping + binning + agg → [grouping, line, stacked bar]</td>
<td>grouping + binning + agg → [grouping, line, stacked bar]</td>
<td>grouping + binning + agg → [grouping, scatter]</td>
</tr>
</tbody>
</table>

Figure 6: Sample SQL and VIS trees
Figure 7: Sample visualizations from TPC-H/TPC-DS

tained 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 because (a) shows too many slices and is hard to visualize; (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 Q08 and Q09, respectively, and (c) and (d) are transformed from TPC-DS Q04 and Q07, 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 $\Delta_{t_i}$, i.e., deletions $\Delta_{t_i}$ and insertions $\Delta_{t_i}^+$. 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 NL4Dv2 [49] 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 binning tree nodes.

**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 $\Delta^+$, 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.

### 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, SQ) pairs on these databases. These (NL, SQ) 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, SQ) 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.
Table 2: nvBench dataset statistics

<table>
<thead>
<tr>
<th>Coverage</th>
<th># Databases</th>
<th># Tables</th>
<th># Domains</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>153</td>
<td>780</td>
<td>105</td>
</tr>
</tbody>
</table>

Top-5 Domains (#Tables)

<table>
<thead>
<tr>
<th>Sport (62)</th>
<th>Customer (52)</th>
<th>School (41)</th>
<th>Shop (35)</th>
<th>Student (22)</th>
</tr>
</thead>
</table>

The Numbers of Columns and Rows

<table>
<thead>
<tr>
<th>#Cols</th>
<th>Avg (#Cols)</th>
<th>Max (#Cols)</th>
<th>Min (#Cols)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,017</td>
<td>5.26</td>
<td>48</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#Rows</th>
<th>Avg (#Rows)</th>
<th>Max (#Rows)</th>
<th>Min (#Rows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,572</td>
<td>1309.65</td>
<td>183,978</td>
<td>1</td>
</tr>
</tbody>
</table>

The Types of Columns

<table>
<thead>
<tr>
<th>C (%)</th>
<th>T (%)</th>
<th>Q (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2763 (68.78%)</td>
<td>465 (11.58%)</td>
<td>789 (19.64%)</td>
</tr>
</tbody>
</table>

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-fit 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 1.5x 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 S2; (R2) it has two subtrees from S1 in total and meets no more than one rule of S2; (R3) it satisfies at least three rules from S2, less than three subtrees from S1 and without keywords from the set of S3; (R4) it has three subtrees from S1 and meets less than three rules from S2 and without keywords from S3; and (R5) it has no more than one subtree from S1 and meets no rule in S2, but exactly has one keyword from S3.

Next we define the hardness of vis tree t.

- **Easy**: t has no more than one subtree from S1, 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.
T2 are to ask humans to verify that whether our synthesized to expected handwritten evaluation are: (T1) Human Tasks.

3.3 Expert and Crowd Evaluation

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 NL 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.

Diversity of NL Queries. For different NL queries wrt. 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 (BLEU’s output is always a number between 0 and 1, with values closer to 1 representing more similar texts), 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

Table 3: NVBench NL queries vs VIS queries

<table>
<thead>
<tr>
<th>VIS Types</th>
<th>#-VIS</th>
<th>#-(NL VIS)</th>
<th>#-NL VIS</th>
<th>Avg. +-W</th>
<th>Min +W</th>
<th>Avg. BLEU (Pair)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
<td>7247</td>
<td>7520</td>
<td>746</td>
<td>22.56</td>
<td>21.38</td>
<td>0.337</td>
</tr>
<tr>
<td>Pie</td>
<td>520</td>
<td>520</td>
<td>1</td>
<td>22.29</td>
<td>21.76</td>
<td>0.341</td>
</tr>
<tr>
<td>Line</td>
<td>352</td>
<td>135</td>
<td>219</td>
<td>28.28</td>
<td>26.56</td>
<td>0.341</td>
</tr>
<tr>
<td>Scatter</td>
<td>266</td>
<td>266</td>
<td>0</td>
<td>22.56</td>
<td>21.38</td>
<td>0.337</td>
</tr>
<tr>
<td>Stacked Bar</td>
<td>359</td>
<td>359</td>
<td>0</td>
<td>22.56</td>
<td>21.38</td>
<td>0.337</td>
</tr>
<tr>
<td>Grouping Line</td>
<td>72</td>
<td>72</td>
<td>1</td>
<td>22.56</td>
<td>21.38</td>
<td>0.337</td>
</tr>
<tr>
<td>Grouping Scatter</td>
<td>127</td>
<td>127</td>
<td>0</td>
<td>22.56</td>
<td>21.38</td>
<td>0.337</td>
</tr>
</tbody>
</table>

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 NL 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.

Diversity of NL Queries. For different NL queries wrt. 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

Figure 11: Sample questions and answers

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

Task T1. 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].

Task T2. 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.

Example 6. Consider Figure 11 with examples from our real test. In Figure 11-a, 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.

Task T3. Given a VIS object, we ask the participants to provide corresponding handwritten NL queries.
Workers, and (by default 3). The majority voting

...obtained crowd workers failed T3 on our test. Task T3 is performed only by experts, because T3 is harder and...different answer, we will ask more questions, but are capped at

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 (nt, 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 (nt, 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 (nt, vis) pairs in T2 that are rated by both crowd workers and

---

**Figure 12: Inter-rater reliability analysis for T2**

**Figure 13: Expert/crowd evaluation**

For T1 and T2, we randomly sampled ~10% (nt, vis) pairs and their associated tables from NVBench. We additionally added n = 100 high-quality handwritten nt queries by two PhDs, such that it has a mixture of machine-generated and handwritten nt queries. Of course, we did not expose which nt queries are machine-generated and which nt queries are handwritten to the participants. T3 was performed along with T1 and T2, if participants can provide new nt 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 appen3, 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 nt 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

---

3https://appen.com/. Figure Eight—formerly CrowdFlower—is now Appen
We further conduct a case study using COVID-19 data (Section 4.6).

We translate 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 total time for humans to build 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) $[q_1, q_2, \ldots, q_l] \in \mathcal{V}^{\text{NL}}$. The output VIs query is also a sequence of tokens $[y_1, y_2, \ldots, y_k] \in \mathcal{V}^{\text{VIs}}$. Here, $\mathcal{V}^{\text{NL}}$ (resp. $\mathcal{V}^{\text{VIs}}$) is the input (resp. output) vocabulary. For example, in Figure 15, the input sequence is $["\text{``Visualize"}, \text{"pie"}, \text{"Select"}, \text{\ldots}]$ and the output sequence is $[\text{"Visualize"}, \text{"pie"}, \text{\ldots}]$.

**Encoder RNN.** Given an input sequence $n_V = [q_1, \ldots, q_l]$, we concatenate $n_V$ with the database table schema information $A = [a_1, a_2, \ldots, a_n]$. We use $X = [x_1, x_2, \ldots, x_n] = [q_1, q_2, \ldots, q_l, a_1, a_2, \ldots, a_n]$ 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) $H = [h_1, h_2, \ldots, h_n]$.

**Decoder RNN.** The decoder is also an RNN based on LSTM cells with attention mechanism [4] that generates the VIs query $[y_1, y_2, \ldots, y_k]$ 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 $s_t$, the previous tokens, and an attention vector $c_t$.

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 “b00_bar”, then we cannot to map an NL query to it; solving this problem is not the focus of this work.

**4 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.

![Figure 14: Task T3 – user time (seconds)](image)

![Figure 15: The seq2vis Model](image)
DeepEye: DeepEye [36] is a rule-based methodology for creating vs chart pairs. Similar to NLADV [42], it cannot 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 NL2vs (Section 4.3), and work better than non-learning based state-of-the-art approaches such as NLADV 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 vs AST tree perspective, we define “vs tree matching accuracy” to measure whether the seq2vis model can precisely predict the vs AST query based on the vs input. From the vs result perspective, we want to know whether the seq2vis model can predict a vs result that exactly equivalent to the gold result even if the predicted vs 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 vs.

**Tree matching accuracy.** It measures if the predicted vs AST tree exactly matches the ground truth vs AST tree. We define Acc_tree = N_tree / n, where N_tree is the number of generated vs AST trees that are exactly equivalent to the ground truth vs AST tree, and n is the total number of trees.

**Result matching accuracy.** In some cases, the vs2sql system may predict some vs queries with “novel” syntax structures, which leads to the predicted vs query and the ground truth vs query may not be the same but the visualization results of the predicted vs query and the ground truth vs query may be the same. To alleviate this issue, we use visualization vs result matching accuracy to measure whether the predicted visualization vs is the same as the ground truth visualization vs. This metric is computed as Acc_res = N_res / n.

**vis component matching accuracy.** A visualization (vs object) consists of three components: vs types, x/y/z-axis, and data with transformation from a database D. In some cases, the vs system can precisely predict the x/y/z-axis and data parts but fail to predict the vs types. Motivated by this, we propose vs component matching metric to reveal the detailed performance of vs2sql model on each vs component. More concretely, for vs types component, we measure the Visualize part of the vs query (Figure 5); for the x/y/z-axis part, we measure the Select component of the vs query; and for the rest part, we measure the Group, Filter, Order, and Superlative components. We compute vs 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 (nt, vs) 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

---

**Figure 16: The distribution of training and testing set**

Figure 16 show the distribution of the training and testing (nt, vs) 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 vs query. Instead of generating V by a learning-based model, we use a heuristic method to extract values from nt queries and fill them into the right part of the predicted vs 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 vs 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 vs tree matching metrics.

Figure 17(a) shows the vs tree matching accuracy, which shows that seq2vis with attention performs the best and achieves averagely 65.6% vs tree matching accuracy, which matches the state-of-the-art performance of n2sql 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 vs copying, and get the lowest scores on the basic seq2vis. Overall, the average performance of the model is related to the difficulty of the vs query, and the performance is better in easier cases.

Figures 17(c)-(e) show the accuracy of three versions of seq2vis on 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
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 (5). 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 vls component matching accuracy. Overall, the seqv2vls with the attention mechanism performs well in each vls component prediction task. We compute the average performance (denoted as Avg.) of three models on each vls 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 seqv2vls 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 seqv2vls perform best on predicting binning operations and work worst on predicting order operations.

4.4 Comparison with the State of the Art

We use the same test set containing 3990 (nts, vls) pairs to compare the performance between seqv2vls 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 seqv2vls with attention significantly outperforms DeepEye and NL4DV, because it learns from the nvBench instead of relying on the rule-based approaches. More concretely, our seqv2vls with attention can better handle hard and extra hard nt questions, but the other two work poorly on nt 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 nt questions. In summary, the learning-based method–seqv2vls significantly outperforms the state-of-the-art rule-based and semantic parser-based approaches for nl2vts tasks. In particular, the learning-based method has the stronger ability to handle those hard and extra hard queries with join, filter, and nested operations, while the state-of-the-art solutions work very poorly on such cases.

4.5 Effect of Low-rated (nt, vls) Pairs

As reported in Section 3.3, the crowd workers and experts have identified 231 imperfect (nt, vls) 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 (nt, vls) pairs. We first removed all low-rated (nt, vls) pairs, identified by the crowd workers and experts, in the training set, to train three versions of seqv2vls models as the baseline. We then randomly
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 the 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 failed NL query. For the failed NL query, as shown in Figure 19-(b)-(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 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 cross-filter-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].

**NL2VIS Techniques.** Text-to-vis [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]. NLD4V [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]. DeepEve [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, SQ) 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, SQ) 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., nvl2visnch) 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