Natural Language to Visualization by Neural Machine Translation

Yuyu Luo, Nan Tang, Guoliang Li*, Jiawei Tang, Chengliang Chai, Xuedi Qin

1 INTRODUCTION

Natural language interface is a promising interaction paradigm for simplifying the creation of visualizations [32, 43, 52]. If successful, even novices can generate visualizations simply like a Google search. Not surprisingly, both commercial vendors (e.g., Tableau’s Ask Data [46], Power BI [2], ThoughtSpot [3], and Amazon’s QuickSight [1]) and academic researchers [7, 13, 20, 33, 34, 40, 42, 45, 49, 50, 57] have investigated to support the translation from NL queries to visualizations (NL2VIS).

NL2VIS needs both natural language understanding that uses machines to comprehend natural language queries, and translation algorithms to generate targeted visualizations using a visualization language. Natural language understanding is considered an AI-hard problem [56], with many intrinsic difficulties such as ambiguity and underspecification. Many tools from the NLP community, especially based on statistical phrase-based translation [26] and neural machine translation [4, 10], have been used to tackle NL2VIS.

The state-of-the-art NL2VIS methods (for example, NL4DV [40] and FlowSense [57]) are statistical phrase-based translation, which treats natural language understanding and machine translation as two steps. They first employ NLP toolkits (for example, NLTK [5], Stanford CoreNLP [37], and NER [12]) to parse an NL query and produce a variety of linguistic annotations (for example, parts of speech, named entities, etc), based on which they then devise algorithms to generate target visualizations. They are good choices when there are not many training datasets to train deep learning models.

We present ncNet, an end-to-end solution using a Transformer-based sequence-to-sequence (seq2seq) model, which translates an NL query to a visualization. It adopts self-attention to generate a rich repre-

Fig. 1: We present ncNet, a Transformer-based sequence-to-sequence model that translates natural language queries to visualizations. It works in two modes. (A) It takes a natural language query $N_1$ and a dataset $D$ as input, translates them $(N_1, D)$ into a visualization rendered in Vega-Lite. (B) Besides a natural language query $N_2$ and a dataset $D$, the user can optionally select a chart template $C$; ncNet will translate the given input $(N_2, C, D)$ into a target visualization.

Abstract—Supporting the translation from natural language (NL) query to visualization (NL2VIS) can simplify the creation of data visualizations because if successful, anyone can generate visualizations by their natural language from the tabular data. The state-of-the-art NL2VIS approaches (e.g., NL4DV and FlowSense) are based on semantic parsers and heuristic algorithms, which are not end-to-end and are not designed for supporting (possibly) complex data transformations. Deep neural network powered neural machine translation models have made great strides in many machine translation tasks, which suggests that they might be viable for NL2VIS as well. In this paper, we present ncNet, a Transformer-based sequence-to-sequence model for supporting NL2VIS, with several novel visualization-aware optimizations, including using attention-forcing to optimize the learning process, and visualization-aware rendering to produce better visualization results. To enhance the capability of machine to comprehend natural language queries, ncNet is also designed to take an optional chart template (e.g., a pie chart or a scatter plot) as an additional input, where the chart template will be served as a constraint to limit what could be visualized. We conducted both quantitative evaluation and user study, showing that ncNet achieves good accuracy in the nvBench benchmark and is easy-to-use.

Index Terms—Natural language interface; data visualization; neural machine translation; chart template;

• Yuyu Luo, Guoliang Li, Chengliang Chai, Xuedi Qin are with the Department of Computer Science, Tsinghua University, China. Email: {luoyi18@mails, liguoliang@, ccl@, qxd17@mails.} tsinghua.edu.cn
• Nan Tang is with QCRI, Hamad Bin Khalifa University, Qatar. Email: ntang@hbku.edu.qa
• Jiawei Tang is with American School of Doha, Doha, Qatar. Email: 23jtang@asd.edu.qa
• *Guoliang Li is the corresponding author.

Manuscript received 21 Mar. 2021; revised 13 June 2021; accepted 8 Aug. 2021.
Date of publication 16 Nov. 2021; date of current version 22 Dec. 2021.
Digital Object Identifier no. 10.1109/TVCG.2021.3114848

1The code is available at https://github.com/Thanksy/ncNet

Authorized licensed use limited to: Tsinghua University. Downloaded on April 25,2023 at 02:27:08 UTC from IEEE Xplore. Restrictions apply.
sentation (high dimensional vectors) of the input. \textit{nclNet} enables smart visualization inference (e.g., guessing the missing column, selecting a chart type, etc).

Besides making smarter inferences, a system can obtain more information (or “hint”) from the user, by either obtaining a one-shot hint from the user or iteratively requiring more information (a.k.a. conversational systems) [6]. The hint can be of various formats, such as NL queries, tables, chart templates, with one main criterion to be easy-to-use. We propose to use chart templates as additional hints, where a user can specify the output to be a pie chart or a scatter plot with a simple click. In practice, chart templates have been widely used in all commercial products, including Tableau, Excel, Google Sheets, and so on. Due to the flexibility of the seq2seq model, we just treat the selected chart template \textit{C} as another sequence, together with the NL query \textit{N} and the dataset \textit{D} as the input \textit{X}.

 Contributions. In this work, we make several contributions, including:

1. proposing \textit{nclNet}, a Transformer-based [53] seq2seq model for supporting NL2VIS;
2. presenting a novel visualization-grammar, namely Vega-Zero, with the main purpose to simplify the NL2VIS translation using neural machine translation techniques. Moreover, transforming it to other visualization languages is straightforward;
3. enhancing \textit{nclNet} by allowing the user to select a chart template, which will be used to improve the translation accuracy;
4. devising two optimization techniques: attention forcing for incorporating pre-defined domain knowledge and visualization-aware translation for better final visualization generation; and
5. demonstrating that \textit{nclNet} can well support NL2VIS with several use cases, as well as conducting a quantitative study.

2 RELATED WORK

2.1 Natural Language Interface for Data Visualization
The idea of using NL as a way to create visualizations was explored around two decades ago [6], where the system interacts with the user through dialogs. During each interaction, the system tries to clarify a small part of the user specification. For example, the system asks: “At what organizational level?”, the user answers: “The idea of using NL”, etc) to train typical deep learning models. (A) Translation from English to French.

An alternative solution is NL2SQL + automatic data visualization, which is a good choice when the entire pipeline is one-shot. However, in practice, it is always iterative. That is, if the target visualization needs to be refined, the user needs to verify/refine both NL2SQ\textit{L} and check the result of automatic data visualization. Note that, checking whether a table is good enough is hard, even for a small table with hundreds/thousands of tuples. In this case, using end-to-end NL2VIS has an advantage that the user only sticks to one task, which is more user-friendly.

3 DESIGN REQUIREMENTS
There are three main goals when devising solutions for NL2VIS, along the same line of other NL2VIS tools e.g., NL4DV [40].

1. \textbf{(1) Easy-to-use.} We want to allow novices to create visualizations simply like a Google-search. That is, even users without data visualization background can easily generate visualizations.

2. \textbf{(2) End-to-end.} Traditional semantic parser based translation systems typically consist of many small sub-components that are tuned separately. In contrast, we want to deliver a complete NL2VIS solution without the need of any additional steps. Besides the well-known benefits of end-to-end solutions such as increased efficiency, cost cutting and ease of learning, one particular benefit for a seq2seq model is that it is easy to maintain and upgrade. For example, upgrading a seq2seq model from using long short-term memory (LSTM) [19] to Transformer [53] only requires to change a few lines of code.

3. \textbf{(3) Language-agnostic.} The main benefit to be language-agnostic is that we just need to train one seq2seq model for NL2VIS, but can support multiple target visualization languages. The practical need for this is evident, because the users might use various visualization languages constrained by different applications, such as Vega-Lite, D3, ggplot2, and so forth.

4 BACKGROUND AND PROBLEM FORMULATION

4.1 Sequence-to-Sequence Models
A sequence-to-sequence (seq2seq) model [51] consists of two parts, an encoder and a decoder, where each part can be implemented by different neural networks. The task of an encoder is to understand the input sequence, and generate a smaller representation \textit{h} (i.e., a high-dimensional vector) to represent the input. The task of a decoder is to generate a sequence of output by taking \textit{h} as input. The network needs to be trained with a lot of training data, in the form of \textit{(Input sequence, Output sequence)} pairs.

Due to the flexibility of seq2seq models that allow the input and output to have different formats, they have a wide spectrum of applications including language translation [17], image captioning [38], conversational models and text summarization [39], and NL to SQL [23].

Let’s first walk through a typical translation task – language translation from English to French (Figure 2(A)). The
4.2 Datasets and Natural Language Query

A natural language query (e.g., “Create a bar chart showing the top 5 states with the most confirmed cases until 2021-03-08”) to a visualization specification (e.g., “le langage naturel à la visualisation est important”).

For example, Figure 3(a) shows a sample data source about the COVID-19 mapping between visual encoding channels and (transformed) data variables. It flattens a JSON object into a sequence of keywords by removing structure-aware symbols such as brackets, colons, and quotation marks. Formally, a unit specification in Vega-Zero is a four-tuple (similar to Vega-Lite but with each tuple being a sequence) as:

\[
\text{unit} = (\text{mark}, \text{data}, \text{encoding}, \text{transform})
\]

Intuitively, we can use Vega-Lite [44]. However, it is hard to train a seq2seq model to generate a hierarchical output (e.g., in JSON format such as Vega-Lite). In contrast, it is much easier to train a seq2seq model to generate a sequence output.

To this end, we present a visualization grammar by simplifying Vega-Lite, with the main purpose to flatten a hierarchical Vega-Lite specification to a sequence-based specification. That is, Vega-Lite is more user friendly, but Vega-Zero is more seq2seq model friendly.

**Vega-Zero.** Vega-Zero keeps most of the keywords of the Vega-Lite about the mapping between visual encoding channels and (transformed) data variables. It flattens a JSON object into a sequence of keywords by removing structure-aware symbols such as brackets, colons, and quotation marks. Formally, a unit specification in Vega-Zero is a four-tuple (similar to Vega-Lite but with each tuple being a sequence) as:

\[
\text{unit} = (\text{mark}, \text{data}, \text{encoding}, \text{transform})
\]

Naturally, as a simplification of Vega-Lite: mark denotes the chart type, including bar, line, point (for scatter chart), arc (for pie chart); data specifies the source data; encoding contains x/y-axis, aggregate function, and color based on which column; and transform defines some data transformation functions: filter, bin, group, sort, and top-k.

For example, Figures 3(d) and (e) show the Vega-Zero grammar and the Vega-Lite grammar for the target histogram in Figure 3(b), respectively. It shows that Vega-Zero flattens the keywords of the Vega-Lite hierarchical JSON specification into a sequence.

**Vega-Zero is Language-Agnostic.** Although Vega-Zero is a simplification of Vega-Lite, we still consider it as language-agnostic, or an intermediate visualization language, because we remove all language-specific settings such as which color to use (e.g., black or blue), what is the default width of a bar, and so on.

Converting a Vega-Zero query into the format of other visualization languages (e.g., Vega-Lite and ggplot2) is straightforward (see Section 5.5).

4.4 Chart Templates

**Chart Templates.** Figure 4 showcases seven chart templates. Each chart template constrains the chart type, the data types of the x/y-axis, and optional sorting parameters.

For example, if a user selects a “Bar Chart” and then sets the sorting parameter as “by measure attributes in descending order”, it means that the user wants to visualize a bar chart and display the bars from high to low, e.g., Figure 3(b).

Essentially, the chart template is used as a constraint to reduce the search space of possible outputs. More specifically, a chart template is selected to explicitly constrain the mark and sort part of a Vega-Zero.

For example, Figure 5(a) shows an “empty” Vega-Zero template without any constraints w.r.t. the type of chart, the dataset to use, how to encode, and how to transform. Assume that a bar chart is selected together with a source CSV file “US_States.csv” and sorting parameter as “by measure attributes in descending order”, then they can be used to fill the mark as bar, data as “US_States.csv”, and sort as y desc.

4.5 Problem Statement

**Input.** Let N be a user provided NL query, C an optional chart template, and D a source dataset. For deep learning, each input needs to be pre-processed before being fed into a deep learning model, typically in the form of vectors [14]. Let \( Y_{1:m} = \text{embed}(N, C, D) \) be a sequence of m vectors after pre-processing the three inputs N, C and D (see Section 5.2 “Input Embedding” for more details).

**Output.** Let \( Y_{1:m} \) be the target visualization w.r.t. the given input, which is a sequence of m vectors.
Consider the three inputs: the NL query $N$, the selected chart $C$, and the table $D$ as shown in Figure 6(a).

Note that, if the chart template is not specified, we will use the “empty” chart template (see Figure 5(a)) as the input sequence; otherwise, the chart template is selected, we will use a partially specified Vega-Zero specification as the input sequence (e.g., see Figure 5(b)).

**Input Tokenization.** We have three types of inputs. Each one will be converted into a sequence, and the three sequences will be concatenated.

1. $N$: The NL query is naturally a sequence of words, where each word will be treated as a token. We add a special token $\{N\}$ (or $\langle N \rangle$) to denote the start (or end) of an NL sequence. For example, the NL query will be tokenized as:

   $$T_N = [\{N\} \text{ show me the trend of confirmed … in utah } \langle N \rangle]$$

2. $C$: Note that there are several tokens such as $[X]$, $[Y]$, $[F]$, $[G]$ (see Figure 5) — these are called masked tokens. The main task of $\text{nclNet}$ is to fill these masked tokens. Moreover, we add a special $\langle C \rangle$ (or $\langle /C \rangle$) token to denote the start (or end) of a chart sequence. For example, the selected line chart is tokenized as:

   $$T_C = [\langle C \rangle \text{ mark line encoding } x [x] y \text{ aggregate … } \langle /C \rangle]$$

3. $D$: For simplicity, we consider a single table, which consists of a set of rows with columns. We linearize the $D$ into a sequence of tokens by concatenating the column names and scanning the table content row by row. Similarly, we add a special $\langle D \rangle$ (or $\langle /D \rangle$) to denote the start (or end) of a table, the columns (or cell values) tokens will be filled between special token $\langle /\text{COL} \rangle$ (or $\langle /\text{VAL} \rangle$) and $\langle /\text{COL} \rangle$ (or $\langle /\text{VAL} \rangle$). For example, the table will be tokenized as:

   $$T_D = [\langle /\text{COL} \rangle \text{ table name } \langle /\text{COL} \rangle \text{ … } \langle /\text{VAL} \rangle \text{ … } \langle /\text{VAL} \rangle \langle /\text{D} \rangle]$$

The above three tokenized sequences will be concatenated as one sequence $X = T_N \oplus T_C \oplus T_D$, which is the input of the “Input Embedding” model, shown in Figure 6(b).

**Input Embedding.** Each token $x_i$ in the input token sequence $X$ will be converted into a vector embedding $\mathbf{x}_i$ as follows:

1. *Token embedding:* Each $x_i$ will be converted to a token embedding, which produces a vector, denoted as $\mathbf{x}_i^\text{token}$.
2. *Type embedding:* Each $x_i$ will also be converted to a type embedding, which is responsible for distinguishing the type of token, e.g., an NL token, a chart template token, or a data token. It also produces a vector, denoted as $\mathbf{x}_i^\text{type}$.
3. *Position embedding:* Finally, we also compute for each $x_i$ a positional embedding, which hints the position of the token $x_i$ in the sequence, denoted as $\mathbf{x}_i^\text{position}$.

Finally, the input embedding of $x_i$, denoted as $\mathbf{x}_i$, is the summation of the above three embeddings as:

$$\mathbf{x}_i = \mathbf{x}_i^\text{token} + \mathbf{x}_i^\text{type} + \mathbf{x}_i^\text{position} = \text{embed}(x_i)$$

After each token is processed as discussed above, it outputs $\mathbf{X}$ as shown in Figure 6(b), i.e., $\mathbf{X} = \text{embed}(N, C, D)$. This $\mathbf{X}$ will serve as the input to the encoder as shown in Figure 6(a).
states

Multi-Head

e.g., up to 512 tokens), it is not allowed to incorporate all values of

To this end, we first measure the similarity between a text value in

typically small enough to be fed to a Transformer.

into the sequence. We propose to only add those values that are

due to the limitation of the length of the input vector of a Transformer

entire

D

NL

C

N

C

D

Encoder

Decoder

nCNet

Vega-Zero

mark line encoding x date y

X

Y

\( encode(h) \)

Output Sequence Generation. Next we describe how the decoder

can auto-regressively generate the output (i.e., the output tokens are

sequentially generated) and thus a mapping of an input sequence \( X \)

to an output sequence \( Y \).

First, the input encoding \( h \) together with a special “start-of-sentence”

\( (\text{SOS}) \) vector, i.e., \( y_0 \), is fed to the decoder. The decoder processes the inputs \( h \) and \( y_0 \) to compute the first target vector \( y_1 \). The first target

vector \( y_1 = “\text{mark}” \) is selected.

Next, the decoder now processes both \( y_0 = (\text{SOS}) \) and \( y_1 = “\text{mark}” \) to compute the second target vector \( y_2 \) (e.g., “line”).

The decoder will then process \( y_0, y_1 \) and \( y_2 \), and so on, done in an

auto-regressive fashion, until a special \( (\text{EOS}) \) token is generated as a

termination condition.

By doing so, it generates an output sequence as \( Y = \text{decode}(h) \).

5.3 Training

nCNet is trained using an existing NL2VIS benchmark called nvBench,

which consists of the mapping from thousands of \((N, D)\) pairs to visu-

alizations, such as \((N_1, D)\) and the bar chart in Figure 1. We insert the

chart template information \( C \) into \((N, D)\) as \((N, C, D)\) with two modes:

(1) empty chart template, as shown in Figure 5(a); and (2) selected chart

template, as shown in Figure 5(b). In both cases, they are uniformly treated as partially completed visualization specifications with many

masked tokens, such as \([X], [Y], [F], \) and \([G] \), to be filled.

The training is to optimize a reconstruction loss – the cross-entropy

between the model output (i.e., the predicted visualization) and the
ground truth visualization provided by the benchmark.

5.4 Optimization

5.4.1 Attention Forcing

A major innovation of Transformer [53] is attention, which allows the model to focus on the relevant parts of the input sequence as needed, and

thus highly improves the quality of machine translation systems. Next,

let’s first review the concept of attention, followed an optimization

technique, called attention forcing, to further improve the quality.

Let’s Pay “Attention”. Figure 6 shows a Transformer-based en-

coder block, where the “Multiple-Head Attention” module gives the

encoder greater power to estimate the relevance of one token to other

tokens. For simplicity, in the following, we only illustrate “One-Head

Attention” (i.e., one attention unit), and “Multiple-Head Attention” just

repeats the computation of one attention unit multiple times in parallel.

A sample attention matrix is shown in Figure 7(a), which shows the

encoder attention weights of pairs of tokens in the range \([0, 1]\) that

correspond to colors from light chartreuse to dark blue. A light color

(or a dark color) indicates that the corresponding two tokens are less

(or more) correlated.

Next we briefly discuss how an attention matrix is computed (see

Figure 6(c)). Recall the input \( X \) is a sequence of vectors, which forms a

matrix by packing these vectors. The Transformer will learn three

weight matrices: the query weight matrix \( W_Q \), the key weight matrix

\( W_K \) and the value weight matrix \( W_V \). It then computes three matrices

as \( Q = X \times W_Q, \) i.e., a matrix that contains the query (vector represen-
tation of one word in the sequence); \( K = X \times W_K, \) i.e., a matrix with all

the keys (vector representations of all the words in the sequence); and

\( V = X \times W_V \) i.e., a matrix with all the values, which are again

the vector representations of all the words in the sequence, where “\( \times \)”

means matrix multiplication. The three matrices, \( Q, K, \) and \( V, \) will be

fed to the attention unit. The output attention matrix is computed as:

\[
\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{Q \times K^T}{\sqrt{d_k}} \right) \times V,
\]

where \( K^T \) is the transpose of matrix \( K \), \( \sqrt{d_k} \) is to stabilize gradients
during training, and \text{softmax} normalizes the weights to sum to 1. Please refer to the Transformer paper [53] for more details.

Attention Forcing. It can be observed from Figure 7(a) that there

is no need to have an attention value between \([X]\) and cell values

recovered, confirmed, died, and utah, because these cell values should
Although the above process will output only one visualization, it is readily to be extended to provide top-k visualizations, e.g., by ranking different branches in Figure 8, by using the ncNet to output the top-2 tokens: “line” and “scatter,” because the rank-1 token “numbers” is not a column of the dataset D.

Remark: Visualization Recommendation. Although the above process will output only one visualization, it is readily to be extended to provide top-k visualizations, e.g., by ranking different branches in Figure 8, which can be used for visualization recommendation.

We have empirically verified that using this optimization can increase the accuracy of the ncNet by default.

5.4.2 Visualization-aware Translation

In the decoding phase, the model may make false prediction due to various reasons. We detail some reasons as follows. The first reason is the inherent ambiguity of the NL query. The second reason is that the NL2VIS model may work poorly in some situations. For example, when it predicts the encoding part, the tokens should be selected from the column names of the table D instead of the cell names of the D.

To alleviate the above issues, we propose a simple yet effective method, namely visualization-aware translation, that incorporates visualization knowledge (i.e., rules of thumb for visualization) and chart templates to validate the output tokens of the ncNet and correct those incorrect tokens (Figure 6(d)).

Our visualization-aware translation method is a variant of the Beam Search [51], a well-known algorithm for neural machine translation task. In a nutshell, the beam search algorithm [47] maintains a set of k candidate partial outputs and their cumulative probabilities at each step, and finally select k candidates with highest cumulative probabilities.

The basic idea of visualization-aware translation is to maintain the top-k possible tokens predicted by the ncNet at some specific steps, and then use visualization rules and the constraints of the visualization-aware translation to select the “most possible” token from the top-k candidates. More concretely, the specific steps are those steps for predicting the empty parts of the Vega-Zero grammar (e.g., the [T]), [X] in Figure 5(a)). For steps of predicting the following types of tokens, by default we maintain top-5 tokens given by the model, and choose the “best” with the following heuristic rules:

1. For the [T]/[S] tokens: if a chart template is specified, we use the parameters from the chart template as the target outputs; otherwise, we take the top-1 candidate as the output token.
2. For the [X]/[Y]/[Z] tokens: we choose the candidate token with the highest probability under the condition that the t is a column of the dataset D and the column type should suitable for the chart type, e.g., two quantitative columns for scatter chart.
3. For the [G] (or [B]) tokens: for the group (or bin) key, we choose the candidate token with the highest probability and it must satisfy: (i) the t is temporal/categorical (or temporal/quantitative) types, and (ii) the t should not be the tokens from the [Z] part.
4. For the [K] token: we keep the candidate token with the highest probability under the condition that the t is a valid number.

For example, Figure 8 shows an example to demonstrate the working mechanism of the visualization-aware translation algorithm. The algorithm follows heuristic rules mentioned before to process output tokens in those specific steps. For example, for the [T] token, since the token “line” has the highest probability and is a right chart type specified by the chart template, the algorithm keeps it as the target output. For the [Y] token, it selects the rank-2 token “number” because the rank-1 token “numbers” is not a column of the dataset D.
5.5 Language-aware Rendering

As shown in Figure 6(e), the Vega-Zero specification can be converted to popular visualization languages (e.g., Vega-Lite, ggplot2) for rendering the visualization result. The translation from a Vega-Zero specification to a targeted visualization language is hard-coded. Currently, we write ∼240 and ∼333 lines of Python code to support the transformation from Vega-Zero to Vega-Lite and ggplot2, respectively. The code is available at https://github.com/Thanksyy/Vega-Zero.

6  IMPLEMENTATION

6.1 Implementation Details of ncNet

We implement the ncNet with PyTorch [41], with three Transformer-based encoder blocks and three Transformer-based decoder blocks. For the multi-head attention layer, we set the number of heads to 8. The embedding dimension for word embedding, token types embedding and position embedding are set to 256. We limit the input length to 512 tokens – inputs that exceed this limit are truncated.

We use the Adam optimizer [24] with a static learning rate (i.e., 0.0005) instead of the one with warm-up and cool-down steps. We use a learned positional encoding. We set the dropout rate as 0.1, for both word embedding, token types embedding and position embedding dimension for word embedding, token types embedding are set to 256. We limit the input length to 512 tokens – inputs that exceed this limit are truncated.

6.2 NL2VIS Benchmark

For training and testing ncNet, we use a public benchmark for the NL2VIS task, namely nvBench [30, 35].

nvBench Statistics. Figure 9 overviews the statistics of nvBench from the data sources and (NL, VIS) perspectives. As shown in Figure 9(a), nvBench has 153 databases along with 780 tables in total and covers 105 domains (e.g., sports, customers). The average number of column/rows of the 780 tables is 5.26/1,309.65, and the maximum/minimum number of columns (rows) is 48/2 (183,978/1). Among the columns, 68.78% of columns are categorical columns, 11.58% of columns are temporal columns, and 19.64% of columns are quantitative columns. Figure 9(a) also depicts the distributions of columns and rows, which tells us that most of the tables have 2 to 9 columns. Given 153 databases, as shown in Figure 9(b), nvBench contains 7,274 visualizations on seven types of charts. For each visualization, nvBench provides one to several NL queries because different users might provide different NL queries for the same visualization. In total, nvBench consists of 25,750 (NL, VIS) pairs. nvBench further defines the four-level complexities, i.e., easy, medium, hard, and extra hard, of the visualizations based on the hardness of the visualization query (i.e., similar to the Vega-Zero). For example, a Vega-Zero with filter, bin and aggregations may be categorized as a Hard visualization. The heatmap in Figure 9(b) shows the distribution of visualizations in different chart types and hardness of visualization queries.

7  EVALUATION

7.1 Usage Scenarios

We first present two common usage scenarios to demonstrate how developers (or lay users) can use ncNet.

ncNet in Jupyter Lab. Data science practitioners often perform interactive data visualization in Jupyter Lab (or Jupyter Notebook). To make ncNet easy-to-use for this type of users, we have developed a Python package to be used in the Jupyter Lab ecosystem.

Figure 11 is a screenshot that demonstrates how to create a desired grouping line chart with an NL query and a chart template for a COVID-19 dataset. First, the user imports the ncNet package and
We conducted a user study on ncNet to test whether it works well in real-world datasets with users from multiple domains? (Figure 11). Alternatively, the user can directly pass the N and C variables to the function ncNet:input_interface(ncquery, chart_template). Finally, ncNet will render the best inferred visualization relevant to N and C in Jupyter Lab using Vega-Lite library (Figure 11).

ncNet as a Toolkit. In addition to using ncNet in an interactive Python environment, we also regard ncNet as a Python toolkit for the NL2VIS task. It means developers can build their visualization systems or dashboards [28, 29, 31, 36] via using ncNet package through pre-defined APIs.

7.2 User Study

We conducted a user study on ncNet with two goals: (1) whether ncNet works well in real-world datasets with users from multiple domains? and (2) collect qualitative feedback on ncNet’s design criterion.

Tasks. To achieve the above goals, we design the following tasks.

Task 1: Given a table and a target visualization, we asked the participants to express an NL query for creating such a visualization.

Task 2: We asked participants to provide their comments about ncNet.

Participants. We invited 4 experts (2 VISers, 1 NLPer, and 1 DBer) and 2 non-experts (1 financial staff and 1 operational staff) to participate in our user study. The four experts have expertise in data analysis/visualization and have more than six years of experience in Python, SQL, and Tableau. The latter two users are familiar with Excel and creating simple visualizations (i.e., plotting two columns as a bar chart in Excel).

Procedures. We asked participants to vote for 5 databases with domains they are familiar with out of 153 databases in nvBench. The 5 databases selected by majority voting were used in the user study. We then taught participants to use ncNet in Jupyter Lab. The experts took about 5 minutes to understand how to use ncNet, while the two non-technical participants took about 20 minutes to understand the usage of ncNet. Non-technical participants had longer learning time mainly because they need to be familiar with the Jupyter Lab environment, including inputting/rephrasing input (NL queries) and executing cell. Third, we randomly sampled 10 visualizations for each database as the targets. Totally, we have 50 target visualizations for the user study. For task 1, we provided descriptions (i.e., what is expected) for each target visualization to guide the participants to formulate their NL queries (i.e., how). We recorded the interaction logs, including NL queries, input records, and task situations, when the participants interacted with ncNet. For task 2, we collected their comments about ncNet.

Results. Figure 12(a) samples 4 target visualizations with user-provided NL queries. We also denote (✓) which NL queries can successfully generate the target visualization using ncNet. Figure 12(b) reports the average accuracy; we can see that the experts achieve better results than non-experts. By rephrasing NL query several times, the accuracy rate is also improved. We also analyze the interaction logs from Task 1. Figure 12(c) shows the time for non-experts to perform a NL2VIS task is about twice than experts. We report more details in Figure 12(d). It depicts the composition of a round of user time, in which thinking time and typing time account for almost half, and the system response time is around 1 second.

The participants comment that that ncNet is easy-to-use and can generate the target visualization in easily. Specifically, the two non-technical users think that the NL2VIS tool is easier than Excel because it only needs to express the intent of visualization using natural language queries instead of conducting a series of operations in the Excel.

7.3 Quantitative Evaluation

We now present quantitative assessments of ncNet using the nvBench testing dataset mentioned in Section 6.3. We use accuracy as the evaluation metrics. To be conservative, the accuracy measures whether the output Vega-Zero sequence exactly matches the ground truth Vega-Zero sequence. We define the accuracy = M/N, where M is the number of the output Vega-Zero sequences that are equivalent to the ground truth Vega-Zero sequences, and N is the total number of testing cases.

We evaluate the following three cases on the same testing dataset:

1. Neural Network-based Baseline: we adapt the state-of-the-art NL2SQL model [18] for NL2VIS task (Figure 13(a)).
2. ncNet without chart template (Figure 13(b)).
3. ncNet with a selected chart template (Figure 13(c)).

Note that, we only consider the first visualization result (i.e., rank-1) outputted by baseline model and ncNet. Figure 13 summarizes the evaluation results. More concretely, the heatmap details the accuracy under different visualization types and difficult levels of the NL query. The bar charts colored in red report the average accuracy by chart types, while the green bar charts show the average accuracy across different hardness of NL queries. Next, we will elaborate the evaluation results.

ncNet: Overall Performance. Figures 13(b) and (c) summarize the performance of ncNet without and with chart template, respectively. The ncNet performs well on different hardness levels of visualization queries (shown in the green bar chart) and works effectively on different types of visualizations (the red bar chart). More specifically, as shown in the heatmaps of Figures 13(b) and (c), we can see that ncNet achieves 100% accuracy on some cases (e.g., (5, Hard) in Figure 13(b) and (c)). Overall, ncNet shows its effectiveness by achieving the accuracy of 77.8% and 79.6% on average for the cases without and with a chart template, respectively.

ncNet: With/Without Chart Template. Next, we give a closer look at the performance difference between two cases of ncNet with and without chart template in Figure 13(b) and (c), respectively. In general, it shows that the accuracy of nearly all cases is higher when an
There are four main causes of evaluation error of ncNet heatmaps (Figure 13). It shows that ncNet outperform it by 12% accuracy, while our approaches outperform it by 12.8% and 14.6%, respectively. More concretely, we can compare the performances across different solutions in the heatmaps (Figure 13). It shows that ncNet is better than the neural network-based baseline in almost all cases. The result is expected, because ncNet uses several optimization techniques specifically designed for the task of NL2VIS.

**Error Analysis.** We conduct an error analysis on those failed cases. There are four main causes of evaluation error of ncNet.

1. **Fail on predicting columns or transform parts.** This is the main cause of failure. For mispredicted columns, it is mainly because the NL query mentions desired columns in an ambiguous or implicit way. In some cases, it fails due to the “unperfect” table header (e.g., “Studl” vs. “Student ID”). Incorporating some pre-trained language models (e.g., BERT [10]) or processing the table headers might help to tackle this issue. The main reason of mispredicting transform parts (e.g., filter) is mainly regard to problem of natural language understanding. That is how to successfully convert the intent into right logic operations. For example, as shown in Figure 12(a), the NL query mentions “… candidates under the age of 45 …”, it should construct the [F] as “… filter Date_of_Birth > 07.01.1976 …”. The above issues could be alleviated by developing more advanced natural language understanding models and designing smart strategies for jointly understanding the user intents incorporating database information and visual analysis knowledge.

2. **Fail on predicting chart type and sorting parameters.** This case usually appears when ncNet does not take a chart template as an additional input.

3. **True result in the rank-k output.** Since our quantitative evaluation only considers the rank-1 Vega-Zero returned by ncNet. However, we observe that the ground truth Vega-Zero can be generated at the rank-k (e.g., rank-4) result by ncNet, in some cases. Therefore, this observation points out that the performance of NL2VIS can be further improved by incorporating visualization recommendation techniques, i.e., suggesting top-k visualization results relevant to the NL query.

4. **Vega-Zero unmatched but visualization result match.** Some of predicted Vega-Zero sequences are unmatched with the ground truth Vega-Zero sequences, but their visualization results are in fact equivalent. For example, “mark bar encoding x cars y aggregate count cars …” equals “mark bar encoding x cars y aggregate count cylinders …” because they have the same aggregate results used for showing the number of cars in a bar chart.

### 8 Limitations and Future Work

#### 8.1 Limitations

**L1) Limited benchmarks.** In retrospect, benchmarks have played a key role in spawning the boom in different research communities, such as ImageNet [9] for image processing, GLUE [55] and SuperGLUE [54] for the NLP community, and TPC benchmarks [16] for the database community. However, large public benchmarks, such as VizNet [21] and nvBench [35], that can be used for deep learning on data visualization tasks are quite limited.

**L2) Supporting only one-shot NL queries.** Different from applications that only need one-shot queries, data analytics often needs to issue a sequence of (or iterative) queries. End-to-end approaches (e.g., NL2VIS) have the advantage that the user only needs to check the visualization and refine the NL query. In contrast, non end-to-end approaches might require the user to be involved in different tasks, which is less user friendly. Even though, however, our current NL2VIS model only translates one (possibly revised) NL to a visualization, instead of translating a sequence of (or conversational) NL pieces into a visualization.

#### 8.2 Future Work

**F1) The quest for more benchmarks.** In response to limitation L1, a promising future direction for pushing deep learning for data visualization, is to contribute to new benchmarks that cover more diversified tasks, such as conversational NL2VIS benchmarks, annotations for chart to description mapping, annotation for similar charts, and so on.

**F2) Supporting conversational NL queries.** To lift limitation L2, an interesting direction is to extend ncNet to support conversational NL queries. The good news is that there are text-to-SQL benchmarks such as CoSQL [58]. Naturally, it is interesting to explore how to leverage these benchmarks to extend and train ncNet such that it can support conversational NL2VIS cases, which have lots of practical applications.

**F3) Char2vec.** Similar to word2vec that learned an universal embedding of words based on the word associations from a large corpus of text, an interesting future work is to learn a universal embedding of charts, so as to enable other downstream applications (e.g., recommendations, story telling, guideline generation, and so forth.)

### 9 Conclusion

We present ncNet, a research attempt using deep neural networks to support NL2VIS. This approach is built upon the latest ILP models, namely Transformer-based seq2seq models. We further propose to use chart templates to help enhance the translation accuracy. We also demonstrate the effectiveness of ncNet on a NL2VIS benchmark over 105 domains. We hope that our proposal ncNet, along with recent advances in ILP, can shed some light on NL2VIS and justify the potential of deep neural networks for NL2VIS. However, the journey just starts and a lot need to be done such as handling non-friendly column header names (or even foo_bar) and work for big data.

### Acknowledgments

This project is supported by NSF of China (61925205, 61632016), Beijing National Research Center for Information Science and Technology (BNRist), Huawei, TAL Education, China National Postdoctoral Program for Innovative Talents (BX2021155), China Postdoctoral Science Foundation (2021M691784), and Zhejiang Lab’s International Talent Fund for Young Professionals.