DeepEye: Towards Automatic Data Visualization

Yuyu Luo[†] Xuedi Qin[†] Nan Tang[‡] Guoliang Li[†]
[†]*Department of Computer Science, Tsinghua University, China* [‡]*Qatar Computing Research Institute, HBKU, Qatar*
{*luoyuyu@mail., qxd17@mails., liguoliang@*}*tsinghua.edu.cn, ntang@hbku.edu.qa*

***Abstract*—Data visualization is invaluable for explaining the significance of data to people who are visually oriented. The central task of automatic data visualization is, given a dataset, to visualize its compelling stories by transforming the data (e.g., selecting attributes, grouping and binning values) and deciding the right type of visualization (e.g., bar or line charts).**

We present DEEPEYE, a novel system for automatic data visualization that tackles three problems: (1) *Visualization recognition:* given a visualization, is it "good" or "bad"? (2) *Visualization ranking:* given two visualizations, which one is "better"? And (3) *Visualization selection:* given a dataset, how to find top-k visualizations? DEEPEYE addresses (1) by training a binary classifier to decide whether a particular visualization is good or bad. It solves (2) from two perspectives: (i) *Machine learning:* it uses a supervised *learning-to-rank model* to rank visualizations; and (ii) *Expert rules:* it relies on experts' knowledge to specify partial orders as rules. Moreover, a "boring" dataset may become interesting after data transformations (e.g., binning and grouping), which forms a large search space. We also discuss optimizations to efficiently compute top-k visualizations, for approaching (3). Extensive experiments verify the effectiveness of DEEPEYE.

I. INTRODUCTION

Nowadays, the ability to create good visualizations has shifted from a nice-to-have skill to a must-have skill for all data analysts. Consequently, this high demand has nourished a remarkable series of empirical successes both in industry (e.g., Tableau and Qlik), and in academia (e.g., DeVIL [1], ZQL [2], SeeDB [3], and zenvisage [4]).

The current data visualization tools have allowed users to create good visualizations, *only if* the users know their data well. Ideally, the users need tools to automatically recommend visualizations, so they can simply pick the ones they like. This is hard, if not impossible, since among numerous issues, no consensus has emerged to quantify the goodness of a visualization that captures human perception.

Technically speaking, "interesting" charts can be defined from three angles: (1) *Deviation-based*: a chart that is dramatically different from the other charts (e.g., SeeDB [5]); (2) *Similarity-based*: charts that show similar trends *w.r.t.* a given chart (e.g., zenvisage [4]); and (3) *Perception-based*: charts that can tell compelling stories, from understanding the data, without being compared with other references.

> *"If I had an hour to solve a problem I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions."*
>
> – Albert Einstein

A. scheduled	B. carrier	C. destination city name	D. departure delay (min)	E. arrival delay (min)	F. passengers
01-Jan 00:05	UA	New York	-4	1	193
01-Jan 04:00	AA	Los Angeles	0	-2	204
01-Jan 06:13	MQ	San Francisco	7	-11	96
01-Jan 07:33	OO	Atlanta	11	-2	112
...	...	...	...	...	...

Table I
AN EXCERPT OF FLIGHT DELAY STATISTICS

Although (1) and (2) can be quantified formally, by statistical deviations and correlations, respectively, our 55 minutes thought is to study (3) despite the hardness of quantifying human perception, because one fundamental request from users is just to find eye-catching and informative charts. The bad news is that users have poor choices for (3).

Example 1: Consider a real-world table about *flight delay statistics of Chicago O'Hare International (Jan – Dec, 2015)*, with an excerpt in Table I (https://www.bts.gov). Naturally, the Bureau of Transportation Statistics wants to visualize some valuable insights/stories of the data.

Figure 1 shows sample visualizations DEEPEYE considers for the entire table. Some are from real use cases.

(i) Figure 1(a) is a scatter plot, with x-axis: `D.departure delay`, y-axis: `E.arrival delay`, and plots grouped (and colored) by "`B.carrier`". It shows clearly the arrival delays *w.r.t.* departure delays for different carriers, e.g., the carrier OO is bad due to its long departure and arrival delays.

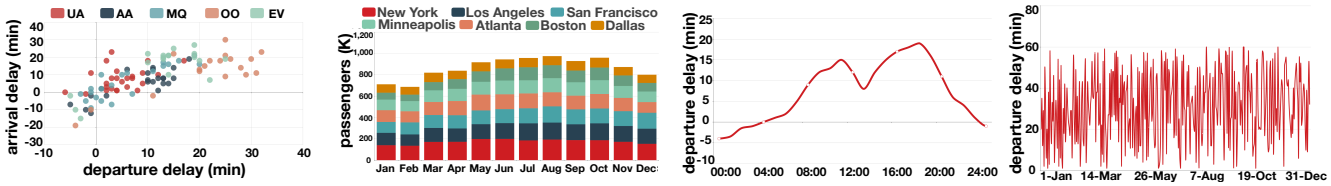
(ii) Figure 1(b) is a stacked bar chart, with x-axis: `A.scheduled` binned by month, y-axis: the aggregated number of `E.passengers` in each month that is further or stacked by `C.destination city`. It shows the number of passengers travelled to *where* and *when*.

(iii) Figure 1(c) is a line chart, with x-axis: `A.scheduled` binned by hour (i.e., the rows with the same hour are in the same bucket), y-axis: the average of `D.departure delay`. It shows *when* is likely to have more departure delays, e.g., it has long delays in late afternoon.

(iv) Figure 1(d) is a line chart, with x-axis: `A.scheduled` binned by date, y-axis: the average of `D.departure delay`. It shows the range of delays, no trend. □

We have conducted a user study with researchers with CS and Visualization background. They all agree that Figures 1(a)–1(c) are *good*, but Figure 1(d) is *bad* because it does not follow any distribution and cannot tell anything.

[1]Guoliang Li is the corresponding author.



(a) Flight delay distribution (b) Monthly #-passengers, by dest. (c) Flight delay *w.r.t.* scheduled time (d) Flight delay *w.r.t.* dates

Figure 1. Sample visualizations for the Flight Delay Statistics table

Problems. DEEPEYE deals with three problems.

1. Visualization Recognition. How to capture human perceptions about whether a visualization is good or bad?

2. Visualization Ranking. Is that possible to rank visualizations to say which one is better?

3. Visualization Selection. In practice, it often needs to show multiple (or top-k) visualizations that, when putting them together, can tell compelling stories of the data at hand.

Challenges. *I. Capturing Human Perception.* How to quantify that which visualization is good, better, or the best?

II. Large Search Space. Sometimes, visualizing a dataset *as-is* cannot produce any interesting output. Appearances can, however, be deceiving, when the stories reside in the data after being transformed, such as selections for columns, groups, and aggregations – these create a huge search space.

III. Lack of Ground Truth. Finding good visualizations is a mining task. Unfortunately, a benchmark or the ground truth of a given dataset is often unavailable.

Intuitively, there are two ways of handling Challenge I: (A) Learning from examples – there are plenty of generic priors to showcase great visualizations. (B) Expert knowledge, e.g., a bar chart with more than 50 bars is clearly bad. Challenge II is a typical database optimization problem that techniques such as pruning and other optimizations can play a role. For Challenge III, fortunately, there are online tables accompanied with well-designed charts, which are treated as good charts. Besides, we also ask researchers to manually annotate to create "ground truth".

Contributions. Our contributions are summarized below.

▷ We approach *Visualization Recognition* by training binary classifiers to determine whether to visualize a given dataset with a specific visualization type is meaningful.

▷ We solve *Visualization Ranking* from two perspectives. We train a supervised *learning-to-rank* model. We also propose to use partial orders (e.g., attribute importance, attribute correlation) such that experts can declaratively specify their domain knowledge. We further propose a hybrid method to combine the rankings from the above two methods.

▷ We tackle *Visualization Selection* by presenting a graph based approach, as well as rule-based optimizations to efficiently compute top-k visualizations by filtering bad visualizations that do not need to be considered.

▷ We conduct experiments using real-world datasets, and visualization use cases, to show that DEEPEYE can efficiently discover interesting visualizations to tell compelling stories.

Organization. Section II formalizes the problems and overviews DEEPEYE. Section III presents ML-based solutions. Section IV describes partial order-based visualization selection. Section V discusses optimizations. Section VI presents empirical results. Section VII discusses related work. Section VIII closes the paper by concluding remarks.

II. OVERVIEW

We first introduce preliminaries (Section II-A), and then define a visualization language to facilitate our discussion (Section II-B). We then overview DEEPEYE (Section II-C).

A. Preliminaries

We consider a relational table D, defined over the scheme $R(A_1, \ldots, A_m)$ with m attributes (or columns).

We study four widely used visualization types: bar charts, line charts, pie charts, and scatter charts.

We consider the following three types of data operations.

1. Transform. It aims to transform the values in a column to new values based on the following operations.

- ***Binning*** partitions the numerical or temporal values into different buckets:
  - *Temporal values* are binned by minute, hour, day, week, month, quarter, year, whose data type can be automatically detected based on the attribute values.
  - *Numerical values* are binned based on consecutive intervals, e.g., $\mathsf{bin1}[0, 10)$, $\mathsf{bin2}[10, 20)$, ...; or the number of targeted bins, e.g., 10 bins.
- ***Grouping*** groups values based on categorical values.

2. Aggregation. Binning and grouping are to categorize data together, which can be consequently interpreted by aggregate operations, `SUM` (sum), `AVG` (average), and `CNT` (count), for the data that falls in the same bin or group. Hence, we consider three aggregation operations: $\mathtt{AGG} = \{\mathtt{SUM}, \mathtt{AVG}, \mathtt{CNT}\}$.

3. Order By. It sorts the values based a specific order. Naturally, we want some scale domain, e.g., x-scale, to be sorted for easy understanding of some trend. Similarly, we can also sort y-scale to get an order on the y-axis.

B. Visualization Language

To facilitate our discussion, we define a simple language to capture all possible visualizations studied in this paper. For simplicity, we first focus on visualizing two columns,

VISUALIZE	TYPE ($\in$ {bar, pie, line, scatter})
SELECT	X', Y' ($X' \in \{X, \texttt{BIN}(X)\}$, $Y' \in \{Y, \texttt{AGG}(Y)\}$)
FROM	D
TRANSFORM	X (using an operator $\in$ {BIN, GROUP})
ORDER BY	X', Y'

Figure 2. Visualization language (two columns)

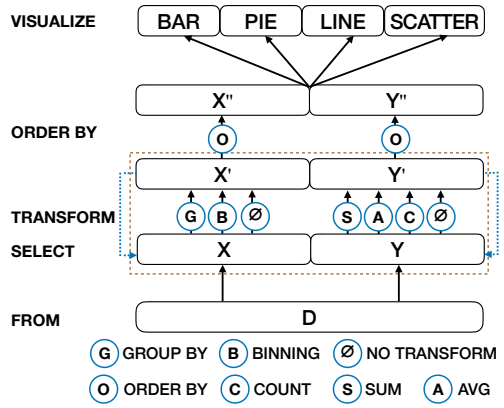


Figure 3. Search space for two columns

as shown in Figure 2. Each query contains three mandatory clauses (**VISUALIZE**, **SELECT**, and **FROM** in bold) and two optional clauses (*TRANSFORM* and *ORDER BY* in italic). They are further explained below.

▷ **VISUALIZE**: specifies the visualization type

▷ **SELECT**: extracts the selected columns

- X'/Y' relates to X/Y: X' is either X or binning values, e.g., by hour; Y' is either Y or the aggregation values (e.g., AGG={SUM, AVG, CNT}) after transforming X

▷ **FROM**: the source table

▷ *TRANSFORM*: transforms the selected columns

- Binning
  - BIN X BY {MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR}.
  - BIN X INTO N, where N is the targeted #-bins.
  - BIN X BY UDF(X), where UDF is a user-defined function, e.g., splitting X by given values (e.g., 0).
- Grouping: GROUP BY X

▷ *ORDER BY*: sorts the selected column, X' or Y'

Example 2: One sample query Q_1 is given below, which is used to visualize Figure 1(c). □

```
Q1 : VISUALIZE line
     SELECT A.scheduled,AVG(D.departure
delay)
     FROM TABLE I
     BIN A.scheduled BY HOUR
     ORDER BY A.scheduled
```

Each query Q over D, denoted by $Q(D)$, will produce a chart, which is also called a *visualization*.

Search Space. Given a dataset D, there exist multiple visualizations. All possible visualizations form our search space, which is shown in Figure 3 for two columns.

▷ **SELECT** can take any ordered column pairs (i.e., XY and YX are different), which gives $m \times (m-1)$.

▷ *TRANSFORM* can either group by X, bin X (we have 9 cases, e.g., by minute, hour, day, week, month, quarter, year, default buckets and UDF), or do nothing; and aggregate Y using different operations. Thus there are $(1+9+1)\times 4 = 44$ cases for each column pair.

▷ *ORDER BY* can order either column X', column Y', or neither: these give 3 possibilities. Note that we cannot sort both columns at the same time.

Together with the four visualization types, the number of all possible visualizations for two columns is: $m \times (m-1) \times 44 \times 4 \times 3 = 528\ m(m-1)$, which is fairly large for wide tables (i.e., the number of columns m is large).

Remark. As surveyed by [6], real users strongly prefer bar, line, and pie charts. In particular, the percentages of bar, line and pie charts are 34%, 23%, and 13% respectively; and the total percentage of the three types is around 70%. Thus this work focuses on these chart types and leaves supporting other chart types as a future work.

Extensions for One Column and Multiple Columns. Our techniques can be easily extended to support one column and multiple columns. For one column, we can do group/bin on the column. In this case, CNT can be applied for the data falling into the same group/bin. So there are $(1+9+1)\times 2 = 22$ cases for transformation. Also, *ORDER BY* can work either on X', on Y', or does not sort any column. Hence, the search space for one column is $m \times 22 \times 4 \times 3 = 264\ m$.

For multiple columns, there are two cases. (i) There are one column X on x-axis, and multiple columns $Y_1, \cdots, Y_z$ on y-axis ($2 \le z \le m-1$). The query aims to compare the Y_i columns for $1 \le i \le z$. There are m cases for x-axis, and $\sum_{i=2}^{m-1} \binom{i}{m}$ cases for y-axis. So the search space for this case is $m \times (1+9+1) \times \sum_{i=2}^{m-1} 4^i \times \binom{i}{m} \times 4 \times (1+i+1) = 44m(i+2)\sum_{i=2}^{m-1} 4^i \binom{i}{m}$. (ii) There are three columns X, Y and Z.We first group the data by X, and for each group, do group/bin on Y, which is used for the x-axis. We then calculate SUM, AVG, CNT of the Z data that falls in the same group/bin as the y-axis. There are m^3 cases for column selection. For each selection, there are 44 cases for transformation of Y and Z. Also, we can sort the data by X', Y', Z', or does not sort any column. Thus the search space is $m^3 \times 44 \times 4 \times 4 = 704m^3$.

Such a large search space calls for a system, such as DEEPEYE, that can navigate this search space and automatically select visualizations.

C. An Overview of DEEPEYE

An overview of DEEPEYE is given in Figure 4, which consists of an offline component and an online component.

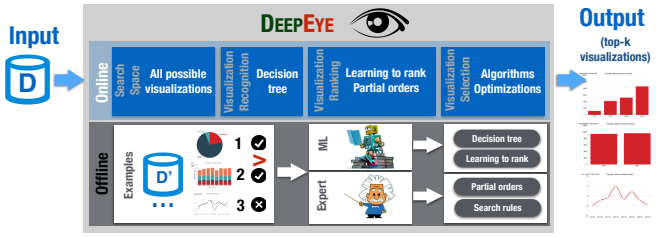


Figure 4. Overview of DEEPEYE

Offline component relies on examples – good visualizations, bad visualizations, and ranks between visualizations – to train two ML models: a binary classifier (e.g., a *decision tree*) to determine whether a given dataset and an associated visualization is good or not, and a *learning-to-rank* model that ranks visualizations (see Section III for more details). This process is done periodically when there are more examples available. Alternatively, experts may specify partial order as rules based on their knowledge to rank visualizations, which will be discussed in Section IV.

Online component identifies all possible visualizations, uses the trained classifier to determine whether a visualization is good or not, employs either the learning-to-rank model or expert provided partial orders to select top-k visualizations (see more details in Sections IV and V).

III. Machine Learning-based Visualization Recognition, Ranking, and Selection

A natural way to capture human perception is by learning from examples. The hypothesis about what are learned from generic priors can be applied to different domains is that the explanatory factors behind the data for visualization is not domain specific, e.g., pie charts are best used when making part-to-whole comparisons (for example, the number of passengers by carrier UA compared to other carriers).

Features. It is known that the performance of machine learning methods is heavily dependent on the choice of features (or representations) on which they are applied. Much of our effort goes into this feature engineering to support effective machine learning. We identify the following features $\mathbf{F}$.

(1) The number of distinct values in column X, $d(X)$.

(2) The number of tuples in column X, $|X|$.

(3) The ratio of unique values in column X, $r(X) = \frac{d(X)}{|X|}$.

(4) The $\max(X)$ and $\min(X)$ values in column X.

(5) The data type $\mathbf{T}(X)$ of column X:

- *Categorical:* contains only certain values, e.g., carriers.
- *Numerical:* contains only numerical values, e.g., delays.
- *Temporal:* contains only temporal data, e.g., date.
  - We also use abbreviations: `Cat` for categorical, `Num` for numerical, and `Tem` for temporal.

(6) The correlation of two columns, $c(X, Y)$, is a value between -1 and 1. The larger the value is, the higher correlation the two columns have. We consider linear, polynomial, power, and log correlations. We take the maximum value in these four cases as the correlation between X and Y.

(7) The visualization type: bar, pie, line, or scatter charts.

For two columns X, Y, we have the above features (1–5) for each column, which gives $6 \times 2 = 12$ features; together with (6) and (7), we have a feature vector of 14 features.

Visualization Recognition. The first task is, given a column combination of a dataset and a specified visualization type, to decide whether the output (i.e., the visualization node) is good or bad. Hence, we just need a binary classier, for which we used decision trees [7]. We have also tested Bayes [8] classifier and SVM [9], and the decision tree outperforms SVM and Bayes (see Section VI for empirical comparisons).

Visualization Ranking. The other task is, given two visualization nodes, to decide which one is better, for which we use a *learning-to-rank* [10] model, which is an ML technique for training the model in a ranking task, which has been widely employed in Information Retrieval (IR), Natural Language Processing (NLP), and Data Mining (DM).

Roughly speaking, it is a supervised learning task that takes the input space $\mathcal{X}$ as lists of feature vectors, and $\mathcal{Y}$ the output space consisting of grades (or ranks). The goal is to learn a function $F(\cdot)$ from the training examples, such that given two input vectors $\mathbf{x_1}$ and $\mathbf{x_2}$, it can determine which one is better, $F(\mathbf{x_1})$ or $F(\mathbf{x_2})$. We used the LambdaMART algorithm [11].

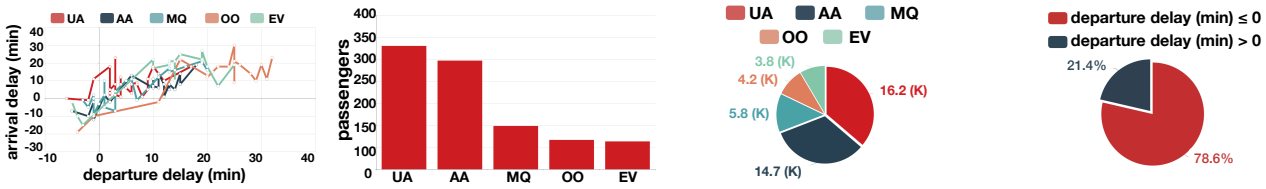
Visualization Selection. Learning to rank model can be used directly for the visualization selection problem: given a set of visualization nodes (and their features vectors) as input, outputs a ranked list.

Remarks. Using ML models as *black-boxes* has two shortcomings. (1) They may not capture human perception as precise as experts in some aspects, e.g., there are not enough examples for comparing visualizations for different columns; and (2) It is hard to improve search performance of black-boxes. Naturally, expert knowledge should be leveraged when it can be explicitly specified.

IV. Partial Order-based Visualization Selection

Computing top-k visualizations requires a *ranking* for all possible visualizations. Ideally, we expect a *total order* of visualizations such that the top-k can be trivially identified. However, it is hard to define a total order, because two visualizations may not be directly comparable. A more feasible way, from the user perspective, is to specify *partial orders* for comparable visualizations. Afterwards, we can obtain a directed graph representing the partially ordered set of visualizations (*a.k.a.* a Hasse diagram).

We first discuss the ranking principle (Section IV-A), and define partial orders (Section IV-B). We then present an algorithm to compute top-k visualizations based on the partial order (Section IV-C). We also propose a hybrid method by combing learning-to-rank and partial order (Section IV-D).



(a) Flight delay distribution (b) Average #-passengers by carriers (c) Total #-passengers *w.r.t.* carriers (d) Proportions of departure delays

Figure 5. More sample visualizations for the Flight Delay Statistics table

A. Visualization Ranking Principle

Definition 1: [Visualization Node.] A *visualization node* consists of the original data X, Y, the transformed data X', Y', features $\mathbf{F}$, and the visualization type $\mathbf{T}$. □

Given two nodes Q_1 and Q_2, we use X_1/Y_1 (resp. X_2/Y_2) to denote the two columns of Q_1 (resp. Q_2), and X_1'/Y_1' (resp. X_2'/Y_2') to denote the transformed columns.

We consider three cases, based on different possibilities of columns shared between two visualizations.

Case 1. $X_1 = X_2$ **and** $Y_1 = Y_2$**:** they have the same original data. Again, we consider two cases, (I) the same transformed data (i.e., $X_1' = X_2'$ and $Y_1' = Y_2'$) and (II) different transformed data ($X_1' \neq X_2'$ or $Y_1' \neq Y_2'$).

(I) $X_1' = X_2'$ and $Y_1' = Y_2'$: we adopt the techniques from the visualization community to rank visualizations [12], [13].

(i) X_1' and X_2' are categorical: pie/bar charts are better than scatter/line charts, because the latter two focus on the trend and correlation between X and Y.
- If Y_1' and Y_2' are obtained by `AVG`, then bar charts are better, because pie charts are best used when making part-to-whole comparisons but we cannot get part-to-whole ratio by the `AVG` operation.
- It would be better to use bar charts if there are many categories (for example, ≥ 10), because it is hard to put many categories in a single pie chart.
- If $\min(Y_1') < 0$, pie charts are not applicable.

(ii) X_1' and X_2' are numerical: scatter/line charts are better than pie/bar charts.
- If there is a correlation between X' and Y', then scatter charts are better, because the scatter plot is simply a set of data points plotted on an x and y axis to represent two sets of variables. The shape those data points create tells the story, most often revealing correlation (positive or negative) in a large amount of data.
- If there is no correlation, line charts are better, because line charts show time-series relationships using continuous data (which is measured and has a value within a range). Line charts allow a quick assessment of acceleration (lines curving upward), deceleration (lines curving downward), and volatility (up/down frequency). Line charts are also excellent for tracking multiple data sets on the same chart to see any correlation in trends.

Observed from Case 1(I), we need to consider a factor to rank different charts. **Factor 1-** *The matching quality between the data and charts*: whether the charts can visualize the inherent features of the data, e.g., trend, correlation.

(II) $X_1' \neq X_2'$ or $Y_1' \neq Y_2'$: they have different transformed data. Typically, the smaller the cardinality of the transformed data, the better.

We consider another factor from Case 1(II). **Factor 2 -** *The quality of transformation operations*: whether the transformation operators make sense.

Case 2: $X_1 \neq X_2$ **or** $Y_1 \neq Y_2$**, and** $\{X_1, Y_1\} \cap \{X_2, Y_2\} \neq \emptyset$**:** They share a common column. Intuitively, for different columns, a user is more interested in visualizing an "important column". We consider another factor based on Case 2. **Factor 3 -** *The importance of a column:* whether it is important to visualize.

Case 3: $\{X_1, Y_1\} \cap \{X_2, Y_2\} = \emptyset$**:** they do not share common attributes. It is hard to directly compare two visualizations. Our hope is to use the transitivity of partial orders, based on the above three factors, to rank them.

B. Partial Order

Now we are ready to formally introduce our methodology to quantify visualizations so as to (partially) rank them, based on the above factors. Let v be a visualization node.

Factor 1: *The matching quality between data and chart* $\mathbf{M}(v)$. It is to quantify the "goodness" of this visualization for the data and visualization type in v, with four cases.

(i) **Pie Chart.** If the aggregation function is `AVG`, i.e., $Y' =$ `AVG`(Y), then the pie chart doesn't make sense as pie charts are best used when making part-to-whole comparisons, and we set the value as 0. If there is only one distinct value $|d(X)| = 1$, we cannot get much information from the pie chart and thus we set the value as 0. If there are a small number of values, the pie chart has large significance, and we set the value as 1. If there are many distinct values (e.g., >10), the significance of the pie chart will decrease [14], and we set the value as $\frac{10}{|d(X)|}$. In addition, if Y values are similar, the pie chart has no much meaning, and we prefer the Y values have large difference. It is defined below.

$$\mathbf{M}(v) = \begin{cases} 0 & \begin{array}{l} |d(X)| = 1 \\ \text{or } \min(Y') < 0 \\ \text{or } Y' = \texttt{AVG(Y)} \end{array} \\ \sum_{y \in Y} -p(y)\log(p(y)) & 2 \leq |d(X)| \leq 10 \\ \frac{10}{|d(X)|} \sum_{y \in Y} -p(y)\log(p(y)) & |d(X)| > 10 \end{cases} \quad (1)$$

(ii) **Bar Chart.** The significance of bar chart is similar to the pie chart and the difference is that bar charts can tolerate large $|d(X)|$ (e.g., >20) [13] and has no requirement that Y values have diverse values, and compute the score as below.

$$\mathbf{M}(v) = \begin{cases} 0 & |d(X)| = 1 \\ 1 & 2 \leq |d(X)| \leq 20 \\ \frac{20}{|d(X)|} & |d(X)| > 20 \end{cases} \quad (2)$$

(iii) **Scatter Chart.** We visualize scatter chart only if X, Y are highly correlated. Thus we can set the value as $c(X, Y)$.

$$\mathbf{M}(v) = c(X, Y) \quad (3)$$

(iv) **Line Chart.** We visualize line charts if X is temporal or numerical columns. We want to see the trend of the Y values. Thus we use the trend distribution to

$$\mathbf{M}(v) = \mathsf{Trend}(Y) \quad (4)$$

where $\mathsf{Trend}(Y) = 1$ if Y follows a distribution, e.g., linear distribution, power low distribution, log distribution or exponential distribution; otherwise, $\mathsf{Trend}(Y) = 0$.

Normalized Significance. Since it is hard to compare the significance of different charts, we normalize the significance for each chart and compute the score as below.

$$\mathbf{M}(v) = \frac{\mathbf{M}(v)}{\mathsf{max}\mathbf{M}} \quad (5)$$

where max$\mathbf{M}$ is the maximal score among all the nodes with the same chart with v.

Factor 2: *The quality of transformations* $\mathbf{Q}(v)$. If the transformed data has similar cardinality with the original data, then the transformation is bad. Thus we use the ratio of the cardinality of the transformed data to the cardinality of the original data to evaluate the quality, i.e., $\frac{|X'|}{|X|}$, and the smaller the better. Thus we compute the value as:

$$\mathbf{Q}(v) = 1 - \frac{|X'|}{|X|} \quad (6)$$

Factor 3: *The importance of columns* $\mathbf{W}(v)$. We first define the importance of a column $X, \mathbf{W}(X)$, which is the ratio of the number of valid charts (those candidate charts) containing column X to the number of valid charts. Clearly, the more important a column is, the better to visualize the chart with the column. Thus we compute the node weight by summing the weight of all columns in the node.

$$\mathbf{W}(v) = \sum_{X \in v} \mathbf{W}(X) \quad (7)$$

We normalize $\mathbf{W}(v)$ into $[0, 1]$ as below.

$$\mathbf{W}(v) = \frac{\mathbf{W}(v)}{max\mathbf{W}} \quad (8)$$

where $max\mathbf{W}$ is the maximal $\mathbf{W}(v)$ among all nodes.

Example 3: For the data in Table 1, we get 44 valid charts after visualization recognition. There are 27 valid charts containing column *scheduled*, and 12 valid charts contain column *departure delay*. So the $\mathbf{W}(v)$ of visualization node Figure 1(c) is $\frac{27}{44} + \frac{12}{44} = 0.89$. □

Given two nodes u, v, if u is better than v on every factor, i.e., $\mathbf{M}(u) \geq \mathbf{M}(v)$, $\mathbf{Q}(u) \geq \mathbf{Q}(v)$, $\mathbf{W}(u) \geq \mathbf{W}(v)$, then intuitively, u should be better than v. Based on this observation, we define a partial order.

Definition 2: [Partial Order] A visualization node u is better than a node v, denoted by $u \succeq v$, if $\mathbf{M}(u) \geq \mathbf{M}(v)$, $\mathbf{Q}(u) \geq \mathbf{Q}(v)$, $\mathbf{W}(u) \geq \mathbf{W}(v)$. Moreover, u is strictly better than v, denoted by $u \succ v$, if any of the above "$\geq$" is "$>$". □

Example 4: Figure 5 shows more visualizations of Flight Delay. We take 2 visualizations in Figure 1 and 3 in Figure 5 to illustrate the *definition of visualization node*, which are shown in Table II. Based on the visualization node in Table II, we can calculate the $\mathbf{M}(v)$, $\mathbf{Q}(v)$ and $\mathbf{W}(v)$ and get Figure 6, which shows the score of three factors that influence partial order of the visualization nodes. And we can get the partial order of the five visualization nodes by Figure 6, which is shown in Figure 7. □

Note that, comparing different types of charts is a hard problem. However, it is common in many search engines, e.g., Google returns ranked results with a mixture of videos, images and webpages. Consequently, any metric is heuristic. As will be verified empirically in Section VI, our normalized scores for different types of charts perform well in practice.

C. Partial Order-Based Visualization Selection

Given a table, we first enumerate all visualizations, and use the trained binary classifier to decide the "valid" charts (i.e., visualizations). Then for every pair of valid charts, we check whether they conform to the partial order. If yes, we add a directed edge. Thus we get a graph $\mathbf{G}(V, E)$, where V is all valid visualization nodes and E indicates visualization pairs that satisfy partial orders. The weight between u and v, where $u \succeq v$, is defined as:

$$\frac{\mathbf{M}(u) - \mathbf{M}(v) + \mathbf{Q}(u) - \mathbf{Q}(v) + \mathbf{W}(u) - \mathbf{W}(v)}{3} \quad (9)$$

We illustrate by examples about how to rank visualization nodes based on the graph.

Example 5: In Figure 7, Figure 1(c) $\succ$ Figure 1(d), so there is a directed edge between visualization node $1(c)$ and visualization node $1(d)$. And the weight is $((1.00 - 0) + (0.99976 - 0.99633) + (0.89 - 0.52))/3 = 0.4578$. Based on the partial order in Figure 7, we can construct the graph $\mathbf{G}$ using the visualization nodes Figure 1(c), Figure 1(d), Figure 5(b), Figure 5(c) and Figure 5(d), which is shown in Figure 8. □

Efficiently Construct the Graph G. It is expensive to enumerate every node pair to add the edges. To address this issue, we propose a quick-sort-based algorithm. Given a node v, we partition other nodes into three parts: those better than v ($v^{\prec}$), those worse than v ($v^{\succ}$), and others ($v^{\not\prec\not\succ}$). Then for each node in $u \in v^{\prec}$ (or $v^{\succ}$), we do not need to compare with nodes in $v^{\succ}$ (or $v^{\prec}$). Thus we can prune many

Fig	$\mathbf{M}(v)$	$\mathbf{Q}(v)$	$\mathbf{W}(v)$
$1(c)$	1.00	0.99976	0.89
$1(d)$	0	0.99633	0.52
$5(b)$	0.73	0.99995	0.36
$5(c)$	1.00	0.99995	0.36
$5(d)$	0.36	0.99998	0.55

Figure 6. Factors of visualization node

Fig	$1(c)$	$1(d)$	$5(b)$	$5(c)$	$5(d)$
$1(c)$	$\succeq$	$\succ$	*none*	*none*	*none*
$1(d)$	*none*	$\succeq$	*none*	*none*	*none*
$5(b)$	*none*	*none*	$\succeq$	*none*	*none*
$5(c)$	*none*	*none*	$\succ$	$\succeq$	*none*
$5(d)$	*none*	$\succ$	*none*	*none*	$\succeq$

Figure 7. Example of partial order

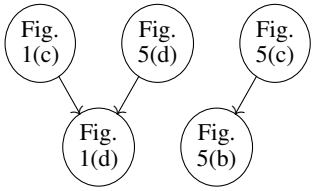


Figure 8. Example of rank visualization

visualization node	attributes		
	data	features	type
Figure 1(c)	$X = scheduled$ $Y = departure\ delay$ $X' =$ BIN(scheduled) BY HOUR $Y' =$ AVG(departure delay)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 24$ $d(X') = 24$ $d(Y') = 18$ $c(X', Y') = 0.43$	line
Figure 1(d)	$X = scheduled$ $Y = departure\ delay$ $X' =$ BIN(scheduled) BY DAY $Y' =$ AVG(departure delay)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 365$ $d(X') = d(Y') = 365$ $c(X', Y') = 0.14$	line
Figure 5(b)	$X = carrier$ $Y = passengers$ $X' =$ GROUP(carrier) $Y' =$ AVG(passengers)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 5$ $d(X') = d(Y') = 5$ $c(X', Y') = N$	bar
Figure 5(c)	$X = carrier$ $Y = passengers$ $X' =$ GROUP(carrier) $Y' =$ SUM(passengers)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 5$ $d(X') = d(Y') = 5$ $c(X', Y') = N$	pie
Figure 5(d)	$X = departure\ delay$ $Y = departure\ delay$ $X' =$ BIN(departure delay) $Y' =$ CNT(departure delay)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 2$ $d(X') = d(Y') = 2$ $c(X', Y') = N$	pie

Table II
EXAMPLE OF VISUALIZATION NODE

unnecessary pairs. We can also utilize the range-tree-based indexing method to efficiently construct the graph [15].

Rank Visualization Nodes based on G. A straightforward method uses topology sorting to get an order of the nodes. It first selects the node with the least number of in-edges, and take it as the best node. Then it removes the node and selects the next node with the least number of in-edges. Iteratively, we can get an order.

However this method does not consider the weights on the edges. To address this issue, we propose a weight-aware approach. We first assign each node with a score $S(v)$.
(1) If node v without out-edge, $S(v) = 0$.
(2) $S(v) = \sum_{(v,u)\in E}(w(v,u)+S(u))$, where $w(v,u)$ is the weight of edge (v,u).

Afterwards, we can select the k nodes with the largest scores. Algorithm 1 shows the pseudo code.

Example 6: We use Figure 8 to illustrate this process. Suppose we want to get the top-3 visualization nodes in this case. Figure 8 shows the graph constructed by the visualization nodes in Figure 7. The out-edges of 5(b) and 1(d) are 0, so the score of 5(b) and 1(d) are 0.

The weights of edges are: $w(1(c), 1(d))$ = 0.4578, $w(5(d), 1(d))$ = 0.1312, $w(5(c), 5(b))$ = 0.09.

The scores of the visualization nodes are:

S(1(c)) = w(1(c), 1(d)) +S(1(d)) = 0.4578,
S(5(d))= w(5(d),1(d))+S(1(d)) = 0.1312,
S(5(c)) = w(5(c), 5(b)) +S(5(b)) = 0.09.

The top-3 visualization nodes are 1(c), 5(d), and 5(c). □

Algorithm 1: Partial Order-Based Selection

Input: $V = \{v_1, v_2, ..., v_n\}$;
Output: Top-k visualization nodes;
for *each node* $v \in V$ **do**
  Compute $M(v)$, $Q(v)$, $W(v)$;
  Partition $V - \{v\}$ into three parts: $V^{\prec}$, $V^{\succ}$, $V^{\not\prec\not\succ}$;
  Prune unnecessary pairs according to partitions;
Construct $\mathbf{G}(V, E)$ based on range-tree-based indexing;
ComputeNodeScore(v = root of V);
return *Top-k nodes* v *with largest weights* $S(v)$*;*

Function ComputeNodeScore

Input : v
Output: $S(v)$
if $outdegree(v) = 0$ **then**
  return $S(v) = 0$
else
  for $(v, u) \in E$ **do**
    ComputeNodeScore(u);
  return $S(v) = \sum_{(v,u)\in E}(w(v,u) + S(u))$*;*

D. Hybrid Ranking Method

Learning-to-rank works well when there are sufficient good examples (i.e., supervised). Partial order works well when the experts have enough expertise to specify domain knowledge (i.e., unsupervised). We propose a hybrid method HybridRank to linearly combine these two methods as follows. Consider a visualization v. Suppose its ranking position is l_v by learning-to-rank and its ranking position is p_v by partial order. Then we assigns v with a score of $l_v + \alpha p_v$, where α is the preference weight of the two methods which can be learned by some labelled data, and rank the visualizations by the score.

V. OPTIMIZING PARTIAL ORDER-BASED APPROACH

A closer look at the process of visualization enumeration (i.e., the search space) suggests that some visualizations should not be considered at all – those visualizations that human will never generate or consider, even if they have unlimited budget (or time). In order to directly prune these bad visualizations, we define rules to capture "meaningful" operations (Section V-A). We then present algorithms that utilize these rules to compute top-k visualizations (Section V-B). We close this section by discussing how to generate rules (Section V-C).

A. Decision Rules for Meaningful Visualizations

We are ready to present the rules that can (possibly) generate meaningful visualizations from three perspectives: (1) transformation rules: whether a grouping or binning

operation is useful; (2) sorting rules: whether a column should be sorted; and (3) visualization rules: whether a certain type of visualization is right choice. These rules use the features (or data representations) discussed in Section III.

1. Transformation Rules. We first consider two columns X and Y, and the techniques can be easily extended to support one column or more than 2 columns. Without loss of generality, we assume that X is for x-axis and Y is for y-axis. Next we discuss how to transform X, Y to X', Y', by considering the two transformation operators (GROUP and BIN). We categorize the rules as follows.

(I) X *is categorial:* we can only group X (cannot bin X). After generating the groups, we apply aggregation functions on Y for two cases. (i) If Y is numerical, we can apply an operation in AGG $=$ {AVG, SUM, CNT}. (ii) If Y is not numerical, we can only apply CNT. Thus, we have two rules.

- $\mathbf{T}(X) = \texttt{Cat}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{GROUP(X)}, \texttt{AGG(Y)}$.
- $\mathbf{T}(X) = \texttt{Cat}, \mathbf{T}(\texttt{Y}) \neq \texttt{Num} \rightarrow \texttt{GROUP(X)}, \texttt{CNT(Y)}$.

(II) X *is numerical:* we can only bin X (cannot group X). After generating the buckets, we can apply aggregation functions on Y. (i) If Y is numerical, we can apply an operation in AGG $=$ {AVG, SUM, CNT}. (ii) If Y is not numerical, we can only apply CNT. Thus we have two rules.

- $\mathbf{T}(X) = \texttt{Num}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{BIN(X)}, \texttt{AGG(Y)}$.
- $\mathbf{T}(X) = \texttt{Num}, \mathbf{T}(\texttt{Y}) \neq \texttt{Num} \rightarrow \texttt{BIN(X)}, \texttt{CNT(Y)}$.

(III) X *is temporal:* we can either group or bin X. After generating the groups or buckets, we can apply aggregation functions on Y. (i) If Y is numerical, we can apply an operation in AGG $=$ {AVG, SUM, CNT}. (ii) If Y is not numerical, we can only apply CNT. Thus we have the following rules.

- $\mathbf{T}(X) = \texttt{Tem}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{GROUP/BIN(X)}, \texttt{AGG(Y)}$.
- $\mathbf{T}(X) = \texttt{Tem}, \mathbf{T}(\texttt{Y}) \neq \texttt{Num} \rightarrow \texttt{GROUP/BIN(X)}, \texttt{CNT(Y)}$.

Example 7: Consider Table I. If X = carrier (categorial) and Y = passengers (numerical), we can apply GROUP(carrier), AVG(passengers) and get Figure 5(b). If X = scheduled (temporal) and Y = departure delay (numerical), we can apply BIN(scheduled), AVG(departure delay) and get Figure 1(c). □

2. Sorting Rules. Given two (transformed) columns, we can sort either X or Y. Intuitively, we sort numerical and temporal values in X but cannot sort categorical values. Note we can sort numerical values in Y; otherwise it does not make sense. Thus we get the following rules.

- $\mathbf{T}(X) = \texttt{Num/Tem} \rightarrow \texttt{ORDER BY(X)}$.
- $\mathbf{T}(Y) = \texttt{Num} \rightarrow \texttt{ORDER BY(Y)}$.

Example 8: Based on Figure 1(c), we can sort scheduled (temporal column) and get a trend of average departure delay, which shows average departure delay fluctuates over time. It stands at the first relative high point ∼11:00, after which it starts to decline and rises again and reaches the peak ∼19:00. □

3. Visualization Rules. For Y, it can be a numerical column but cannot be other types of columns.

(I) If X is *categorical*, Y is *numerical*, we can only draw bar charts and pie charts.

(II) If X is *numerical*, Y is *numerical*, we can draw the line charts and bar charts. Moreover, if X, Y have correlations, we can also draw scatter charts.

(III) If X is *temporal*, Y is *numerical*, we draw line charts.

Thus we can get the following rules.

- $\mathbf{T}(X) = \texttt{Cat}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{bar/pie}$.
- $\mathbf{T}(X) = \texttt{Num}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{line/bar}$.
- $\mathbf{T}(X) = \texttt{Num}, \mathbf{T}(\texttt{Y}) = \texttt{Num}$, (X, Y) correlated$\rightarrow$scatter.
- $\mathbf{T}(X) = \texttt{Tem}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{line}$.

Example 9: Figure 5(b) is a meaningful bar chart, which consists of categorical column carrier as X and numerical column passengers as Y. □

B. Rule-based Visualization Selection

An Enumeration Algorithm. A straightforward algorithm enumerates every column pairs. (We need to consider both (X, Y) and (Y, X).) For each pair (X, Y), we enumerate every transformation rule. If the rule can be applied, we transform the data in the two columns into (X', Y'). Then we enumerate every sorting rule and transform it into (X'', Y''). Next, we try different visualization rules and draw the charts if the rule can be applied to (X'', Y'').

Based on these rules, we can get a set of visualization candidates. Next we use them to construct a graph and select top-k visualizations from the graph. However, this algorithm is rather expensive as it requires to first enumerate all candidates and then identify top-k ones from the graph. Next we propose optimization techniques.

A Progressive Method. We propose a progressive method to improve the performance of identifying top-k visualizations. The basic idea is that we do not generate all the candidate visualizations. Instead, we progressively generate the candidates with the largest possibility to be in the top-k results.

Algorithm Overview. For each type of column, categorical, temporal, numerical, we keep a list of charts *w.r.t.* the column type, i.e., $\mathcal{L}_c$, $\mathcal{L}_t$, $\mathcal{L}_m$. We progressively generate the lists. For each list, we split it into different sublists based on the columns, we use $\mathcal{L}_c^X$ to denote the list of charts that take the categorical column X as x-axis. We can similarly define $\mathcal{L}_t$, $\mathcal{L}_n$ for temporal and numerical columns. Then we build a tree-like structure. The dummy root has three children $\mathcal{L}_c$, $\mathcal{L}_t$, $\mathcal{L}_m$. Each node $\mathcal{L}_c$ has several children, e.g., $\mathcal{L}_c^X$, for each categorical column X in the table. Next we use the tournament-like algorithm to select the best chart from leaf to root. For leaf nodes, we generate the best visualization in each leaf node *w.r.t.* the partial order. Then for each node $\mathcal{L}_c$, we select the best visualization from the visualizations of its children. Similarly from the root, we can select the best

visualization from its children. If the best chart is selected from $\mathcal{L}_c^X$, we get the next best chart from the list and adjust the tournament. After we get k charts, it terminates.

Computing the best chart from $\mathcal{L}_c^X$ in the leaf node. For each list $\mathcal{L}_c^X$, we can only generate the bar chart and pie chart. We can get a list of charts based on each factor. Then we get the best one from these lists.

Computing the best chart from $\mathcal{L}_n^X$ in the leaf node. For each list $\mathcal{L}_n^X$, we can only generate the line chart and bar chart. We can get a list of charts based on each factor. Then we get the best one from these lists.

Computing the best chart from $\mathcal{L}_t^X$ in the leaf node. For each list $\mathcal{L}_t^X$, we can only generate the scatter chart. We can get a list of charts based on each factor. Then we get the best one from these lists.

Computing the best chart from $\mathcal{L}_c/\mathcal{L}_t/\mathcal{L}_m$. We just need to select the best one from its children.

Computing the best chart from the root. We compare different charts from its children and select the best one.

Based on the tournament we can generate the top-k charts without generating all the candidate charts.

Optimizations. We propose several optimization techniques.

First, for each column X, when grouping and binning the column, we compute the AGG values on other columns together and avoid binning/grouping multiple times.
(1) For each categorical/temporal column, we group the tuples in D and compute the CNT value; for each numerical column, we compute the AVG and SUM values in each group. Next we visualize the data based on the visualization rules.
(2) For each temporal column, we bin the tuples in D, and compute the CNT value; for each numerical column, we compute the AVG and SUM values in each bin. Next we visualize the data based on the visualization rules.
(3) For each numerical column, we bin the tuples in D, and compute the CNT value; for each numerical column, we compute the AVG and SUM values in each group. Next we visualize the data based on the visualization rules.

Second, we do not generate the groups of a column if there have k charts in $\mathcal{L}_c$ better than any chart in this column.

Third, we postpone many operations after selecting the top-k charts, e.g., sorting, AVG operations. Thus we avoid many unnecessary operations that are not in top-k.

C. Rule Generation and Completeness

Below, we will discuss the "completeness" of rules introduced in Section V-A, in terms of that they cover all cases that a visualization can potentially be meaningful (or good).

Transformation Rule Generation and Completeness. For transformation rule, we only need to consider categorical, numerical, and temporal columns. For categorical column, we can only apply group operations on it and apply aggregation on other columns. For numerical and temporal columns, we can only apply bin operations on it and apply aggregation on other columns. We can see that our rules consider all the possible cases and the transformation rules are complete.

#-tuples			#-columns		
Max	Min	Avg	Max	Min	Avg
			Temporal/Categorical/Numerical/All		
99527	3	3381	2/12/21/25	0/0/1/2	1/2/5/7

Table III
STATISTICS OF EXPERIMENTAL DATASETS

Sorting Rule Generation and Completeness. It is trivial to generate sorting rules because we can only sort the numerical and temporal values on x-axis and numerical values on y-axis. We can see that our rules consider all the possible cases and the sorting rules are complete.

Visualization Rule Generation and Completeness. We only need to consider categorical, numerical, and temporal columns. We can only put the numerical columns on y-axis, and put categorical, numerical, and temporal columns on x-axis. For each case, there are four possible charts. Our rules consider all cases and the visualization rules are complete.

VI. EXPERIMENTS

The key questions we answered in this evaluation are: (A) How does DEEPEYE work for real cases? (B) How well does DEEPEYE perform in visualization recognition? (C) Whether the visualization selection of DEEPEYE can well capture human perception? (D) How efficient is DEEPEYE?

Datasets. We have collected 42 real-world datasets from various domain such as real estate, social study, and transportation. Some statistics are given in Table III: the number of tuples ranges from 3 to 99527, with an average 3381; the number of columns is from 2 to 25; the statistics of #-columns for temporal, categorical, numerical is also given.

Ground Truth. We have asked 100 students to label the dataset. (1) For each dataset, we enumerated all the possible candidate visualizations and asked them to label which are good/bad. (2) For good visualizations, we asked them to compare two visualizations which are better. Then we merged the results to get a total order [16], [**?**]. We got 2520/30892 annotated good/bad charts, and 285,236 comparisons for visualization pairs. Note that if a table has k visualizations, there are $k \times (k-1)/2$ rankings for one table.

Training. We selected 32 datasets as training datasets and trained ML models based on the ground truth of 32 datasets. We tested on other 10 datasets – this can help justify whether the trained ML models can be generalized. These 10 tables are given in Table IV, which are selected to cover different domains, various number of tuples and columns. Note that the last column, #-charts, refers to good visualizations. We also conducted cross validation and got similar results.

Experimental Environment. All experiments were conducted on a MacBook Pro with 8 GB 2133 MHz RAM and 2.9 GHz Intel Core i5 CPU, running OS X Version 10.12.3.

No.	name	#-tuples	#-columns	#-charts
X1	*Hollywood's Stories*	75	8	48
X2	*Foreign Visitor Arrivals*	172	4	10
X3	*McDonald's Menu*	263	23	275
X4	*Happiness Rank*	316	12	123
X5	*ZHVI Summary*	1,749	13	36
X6	*NFL Player Statistics*	4,626	25	209
X7	*Airbnb Summary*	6,001	9	42
X8	*Top Baby Names in US*	22,037	6	17
X9	*Adult*	32,561	14	103
X10	*FlyDelay*	99,527	6	44

Table IV
10 TESTING DATASETS

No.	name	from
D1	*Happy Countries*	http://www.kenflerlage.com/2016/08/whats-happiest-country-in-world.html
D2	*US Baby Names*	https://deepsense.io/us-baby-names-data-visualization/
D3	*Flight Statistics*	https://www.transtats.bts.gov/airports.asp?pn=1
D4	*TutorialOfUCB*	https://multimedia.journalism.berkeley.edu/tutorials/data-visualization-basics/
D5	*CPI Statistics*	https://medium.com/towards-data-science/data-visualization
D6	*Healthcare*	https://getdataseed.com/demo/
D7	*Services Statistics*	https://getdataseed.com/demo/
D8	*PPI Statistics*	https://ppi.worldbank.org/visualization/ppi.html
D9	*Average Food Price*	http://data.stats.gov.cn/english/vchart.htm

Table V
9 REAL USE CASES WITH DATA AND VISUALIZATIONS

A. Coverage in Real Use Cases

We used 9 real-world datasets in Table V (different from the above training datasets) with both datasets and widely used charts. The 32 training datasets are used for learning.

Figure 9 is a screenshot of the 1st page (i.e., top-6 results) of running DEEPEYE on D3. This is the best case since all 4 visualizations used by the website are automatically discovered by DEEPEYE in the first page. Note that traditionally, this will take hours for experienced data analysts who know the data very well to produce; now, you blink and it's done.

Applying DEEPEYE for other datasets are shown in Table VI. Take dataset D1 for instance, Table VI shows that D1 has 5 practically used visualizations, which can be covered by top-23 results from DEEPEYE.

We have two main research findings from this group of experiment. (1) DEEPEYE can automatically discover visualizations needed in practice to tell compelling stories, which makes creating good visualizations a truly sexy task. (2) Sometimes the k visualizations needed to cover real cases is much larger than the #-real ones, e.g., it needs top-23 results to cover the 5 real cases. This is not bad at all since (i) users just browse few pages to find the ones they need; (ii) the other results not used by the real cases are not necessarily bad ones, for many cases the users may like them if they have seen them.

Figure 9. Screenshot of DEEPEYE for Dataset D3 Flight Statistics

	Vis	top-k
D1	5	23
D2	5	11
D3	4	6
D4	4	9
D5	1	1
D6	2	3
D7	6	24
D8	9	32
D9	27	27

Table VI
COVERAGE

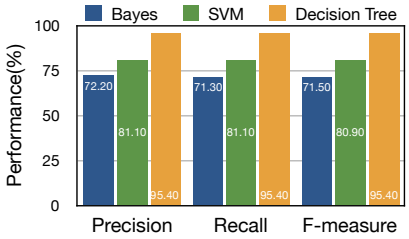


Figure 10. Average effectiveness

	Precision			Recall			F-measure		
	Bayes	SVM	DT	Bayes	SVM	DT	Bayes	SVM	DT
B	84.30	86.90	93.20	84.10	86.40	93.00	84.20	86.65	93.10
L	93.20	96.50	99.50	90.80	96.40	99.50	91.98	96.45	99.50
P	83.40	90.60	94.70	82.60	90.60	94.70	83.00	90.60	94.70
S	84.30	86.90	93.10	84.10	86.40	92.90	84.20	86.65	93.00

Table VII
AVERAGE EFFECTIVENESS (%): B(BAR), L(LINE), P(PIE), S(SCATTER)

	Bar			Line			Pie			Scatter		
	Bayes	SVM	DT	Bayes	SVM	DT	Bayes	SVM	DT	Bayes	SVM	DT
X1	79	81	93	81	83	93	82	86	95	83	83	95
X2	82	91	98	85	90	99	84	90	98	83	90	98
X3	71	80	95	84	92	94	82	84	94	81	82	95
X4	72	82	94	84	91	93	82	87	95	82	84	95
X5	73	83	94	86	89	96	83	86	95	82	83	94
X6	73	80	95	86	87	95	84	84	94	82	83	96
X7	71	83	96	87	90	95	83	85	94	81	82	95
X8	70	81	95	89	86	96	82	84	95	81	83	94
X9	72	82	94	90	88	93	82	84	95	82	82	96
X10	71	81	97	81	86	97	83	83	96	83	84	96

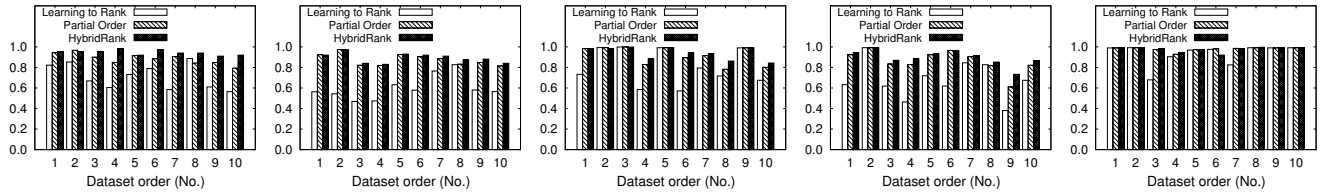
Table VIII
F-MEASURE (%) FOR DIFFERENT TYPES OF CHARTS

B. Visualization Recognition

Our main purpose in this group of experiment is to test (1) whether binary classifiers can well capture human perception for visualization recognition; and (2) which ML model best fits our studied problem?

We tested three popular ML models – Bayes, SVM and decision tree (DT). We used precision, recall and F-measure (i.e., the harmonic mean of precision and recall).

Figure 10 shows the average precision, recall and F-measure values for the 10 datasets (X1–X10). This figure clearly shows that decision tree is way better than SVM and Bayes as binary classifiers for visualization recognition and



(a) NDCG for 10 datasets (b) NDCG for bar charts (c) NDCG for line charts (d) NDCG for pie charts (e) NDCG for scatter charts

Figure 11. Effectiveness study for visualization ranking & selection (x-axis: dataset; y-axis: NDCG)

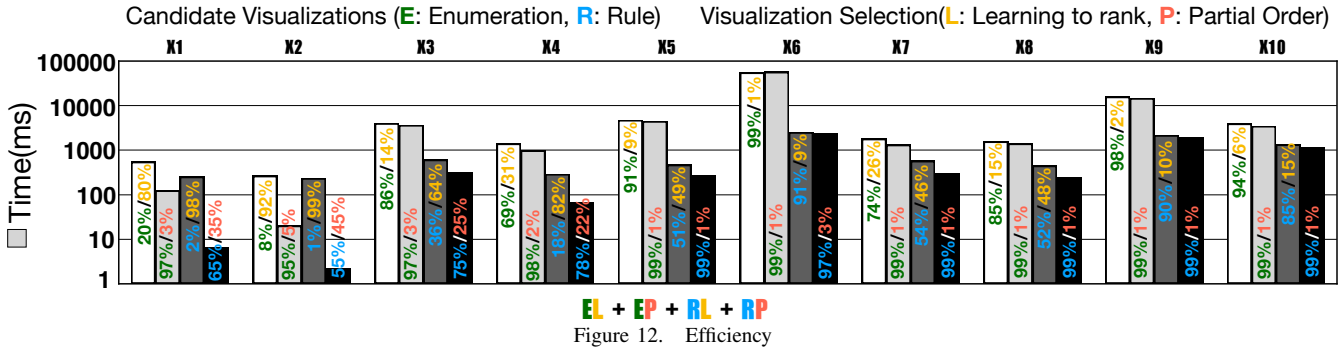


Figure 12. Efficiency

achieves averagely 95% F-measure – this justifies decision tree as a good choice for visualization recognition problem. The main reason is possibly because the visualization recognition should follow the rules as discussed in Section V-A and decision tree could capture these rules well.

Table VII breaks down Figure 10 to show the effectiveness for bar (B), line (L), pie (P), and scatter (S) charts, which is the average of the 10 tested datasets. It shows the consistent story that decision tree outperforms SVM and Bayes behaves the worst. Table VIII further verifies the above results by showing individual cases for these 10 datasets, which also confirms that decision tree works the best.

C. Visualization Selection

We used the normalized discounted cumulative gain (NDCG) [17] as the measure of ranking quality, which calculates the gain of a result based on its position in the result list and normalizes the score to $[0, 1]$ where 1 means perfect top-k results. We compared the NDCG values of partial order-based method and learning to rank for X1–X10.

Figure 11(a) reports the results. It shows clearly that *partial order* is always better than *learning to rank*. The maximal NDCG of *partial order* is 0.97, and minimal NDCG of *partial order* is 0.81, while the maximal and minimal NDCG of *learning to rank* are 0.85 and 0.52, respectively. This is because the *partial order* ranked the order based on expert rules which captures the ranking features very well and *learning to rank* cannot learn these rules. HybridRank outperforms learning-to-rank and partial order. For example, the average NDCG of Hybrid is 0.94 and outperforms learning-to-rank and partial order by 32.4% and 6.8%, respectively.

Figures 11(b), 11(c), 11(d) and 11(e) classify Figure 11(a) into bar, line, pie and scatter charts, respectively. Not surprisedly, they behave differently for various datasets. However, the general observation is that the partial order based approach beats learning to rank for visualization selection.

D. Efficiency – Tell the stories of your data in seconds!

We have also tested the efficiency of DEEPEYE on datasets X1–X10. Each dataset is associated with 4 bars that measure the end-to-end running time from a given dataset to visualization selection. The time of each bar consists of two parts: (i) generate all candidate visualization without/with (i.e., E/R) using our transformation/sorting/visualization rules; and (ii) visualization selection using learning to rank/partial order-based solutions. We annotate the percentage (%) of these two parts in each bar, e.g., the first bar means that it needs 550 ms, where visualization enumeration(E) takes 20% time and visualization selection using learning to rank(L) takes 80%.

Figure 12 tells us the followings: (1) using the rules (Section V-A) can effectively reduce the running time, i.e., RL (resp. RP) runs always faster than EL (resp. EP) since it avoids generating many bad visualizations, as expected; (2) partial order-based approach runs faster than learning to rank model, i.e., EP (resp. RP) runs always faster than EL (resp. RL), because partial order can efficiently prune the bad ones while learning to rank must evaluate every visualizations; (3) DEEPEYE can run to complete in seconds for datasets with reasonable size. Note that the performance will be boosted by DBMSs (e.g., the database-based optimizations in SeeDB [5] and DeVIL [1]) or MapReduce-like platforms such as Spark and Flink since the task of visualization selection is trivially parallelizable.

VII. RELATED WORK

Visualization Recommendation. There has been work on recommending visualizations, such as SeeDB [3], Profiler [18], and Voyager [19]. SeeDB [3] quantifies an "interesting" visualization as the one that is largely deviated from a user given reference, which is similar to find an outlier. Profiler [18] is similar to SeeDB, which findsr anomalies as

candidate recommendations. Voyager [19] suggests visualizations based on statistical properties of all visualizations.

Different from them that mainly use statistical properties (e.g., outliers) for computing recommendations, DEEPEYE tries to capture the human perception by understanding existing examples using mature ML-based techniques.

Interactive Data Visualization Systems. DeVIL [1] employs a SQL-like language to support interactive visualization. zenvisage [4] tries to find other interesting data when the users provide their desired trends, patterns, or insights. Lyra [20] is an interactive environment that enables custom visualization design without writing any code.

DEEPEYE is an automatic visualization system, which is orthogonal to, and can leverage interactive systems.

Data Visualization Languages. There have been several work on defining visualization languages. ggplot [21] is a programming interface for data visualization. The Logical Visualization Plan (LVP) [22] is a nested list of clauses. DeVIL [1] uses a SQL-like language. ZQL [4] borrows the idea Query-by-Example (QBE) that has a tabular structure. Vega (https://vega.github.io/vega/) is a visualization grammar in a JSON format. VizQL [23], used by Tableau, is a visual query language that translates drag-and-drop actions into data queries and then expresses data visually.

Our proposed language is a subset, but shares many features with the others. Our purpose to define a simple language is just to make our discussion easier.

VIII. CONCLUSION AND FUTURE WORK

We have presented DEEPEYE, a novel automatic data visualization system. We leveraged machine learning techniques as black-boxes and expert specified rules, to solve three challenging problems faced by DEEPEYE, namely, visualization recognition, visualization ranking, and visualization selection. We have shown promising results using real-world data and use cases. One major future work is to support keyword queries such that users specify their intent in a natural way [24], [25].

ACKNOWLEDGEMENT

This work was supported by 973 Program of China (2015CB358700), NSF of China (61632016, 61472198, 61521002, 61661166012), and TAL education.

REFERENCES

[1] E. Wu, F. Psallidas, Z. Miao, H. Zhang, and L. Rettig, "Combining design and performance in a data visualization management system," in *CIDR*, 2017.

[2] T. Siddiqui, J. Lee, A. Kim, E. Xue, X. Yu, S. Zou, L. Guo, C. Liu, C. Wang, K. Karahalios, and A. G. Parameswaran, "Fast-forwarding to desired visualizations with zenvisage," in *CIDR*, 2017.

[3] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis, "SEEDB: efficient data-driven visualization recommendations to support visual analytics," *PVLDB*, 2015.

[4] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. G. Parameswaran, "Effortless data exploration with zenvisage: An expressive and interactive visual analytics system," *PVLDB*, 2016.

[5] M. Vartak, S. Madden, A. Parameswaran, and N. Polyzotis, "Seedb: automatically generating query visualizations," *PVLDB*, 2014.

[6] L. Grammel, M. Tory, and M.-A. Storey, "How information visualization novices construct visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 943–952, Nov. 2010.

[7] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, 1986.

[8] I. Rish, "An empirical study of the naive bayes classifier," in *IJCAI*, 2001.

[9] Y. B. Dibike, S. Velickov, D. Solomatine, and M. B. Abbott, "Model induction with support vector machines: introduction and applications," *Journal of Computing in Civil Engineering*, vol. 15, no. 3, pp. 208–216, 2001.

[10] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender, "Learning to rank using gradient descent," in *ICML*, 2005.

[11] C. J. C. Burges, K. M. Svore, Q. Wu, and J. Gao, "Ranking, boosting, and model adaptation," *Technical Report, MSR-TR-2008-109*, 2008.

[12] J. D. Mackinlay, P. Hanrahan, and C. Stolte, "Show me: Automatic presentation for visual analysis," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1137–1144, 2007.

[13] J. D. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, no. 2, pp. 110–141, 1986.

[14] W. S. Cleveland and R. Mcgill, "Graphical perception: Theory, experimentation, and application to the development of graphical methods," vol. 79, pp. 531–554, 09 1984.

[15] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars, *Computational geometry: algorithms and applications*. Springer, 2008.

[16] X. Zhang, G. Li, and J. Feng, "Crowdsourced top-k algorithms: An experimental evaluation," *PVLDB*, 2016.

[17] H. Valizadegan, R. Jin, R. Zhang, and J. Mao, "Learning to rank by optimizing NDCG measure," in *NIPS*, 2009.

[18] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer, "Profiler: integrated statistical analysis and visualization for data quality assessment," in *AVI*, 2012.

[19] K. Wongsuphasawat, D. Moritz, A. Anand, J. D. Mackinlay, B. Howe, and J. Heer, "Voyager: Exploratory analysis via faceted browsing of visualization recommendations," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 649–658, 2016.

[20] A. Satyanarayan and J. Heer, "Lyra: An interactive visualization design environment," *Comput. Graph. Forum*, vol. 33, no. 3, pp. 351–360, 2014.

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

[22] E. Wu, L. Battle, and S. R. Madden, "The case for data visualization management systems," *PVLDB*, 2014.

[23] P. Hanrahan, "Vizql: a language for query, analysis and visualization," in *SIGMOD*, 2006.

[24] X. Qin, Y. Luo, N. Tang, and G. Li, "Deepeye: Visualizing your data by keyword search," in *EDBT Vision*, 2018.

[25] Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang, "Deepeye: Creating good data visualizations by keyword searc," in *SIGMOD demo*, 2018.