Table2Charts: Recommending Charts by Learning Shared Table Representations

Mengyu Zhou*
Microsoft Research

Qingtao Li†
Peking University

Xinyi He†
Xi'an Jiaotong University

Yuejiang Li†
Tsinghua University

Yibo Liu†
New York University

Wei Ji*
Microsoft

Shi Han*
Microsoft Research
Beijing, China

Yining Chen*
Daxin Jiang*
Microsoft

Dongmei Zhang*
Microsoft Research
Beijing, China

ABSTRACT

It is common for people to create different types of charts to explore a multi-dimensional dataset (table). However, to recommend commonly composed charts in real world, one should take the challenges of efficiency, imbalanced data and table context into consideration. In this paper, we propose Table2Charts framework[1] which learns common patterns from a large corpus of (table, charts) pairs. Based on deep Q-learning with copying mechanism and heuristic searching, Table2Charts does table-to-sequence generation, where each sequence follows a chart template. On a large spreadsheet corpus with 165k tables and 266k charts, we show that Table2Charts could learn a shared representation of table fields so that recommendation tasks on different chart types could mutually enhance each other. Table2Charts outperforms other chart recommendation systems in both multi-type task (with doubled recall numbers R@3=0.61 and R@1=0.43) and human evaluations.

CCS CONCEPTS

• **Human-centered computing** → **Visualization**; • **Computing methodologies** → **Machine learning**; *Natural language processing*; • **Information systems** → *Information systems applications.*

KEYWORDS

Table2seq; chart recommendation; deep Q-learning; copying mechanism; search sampling; transfer learning; table representations

*Author emails: {mezho, jiwe, shihan, yinichen, djiang, dongmeiz}@microsoft.com.
†The contributions by Qingtao Li, Xinyi He, Yuejiang Li and Yibo Liu have been conducted and completed during their internships at Microsoft Research Asia, Beijing, China. Their school emails are: newdaylqt@pku.edu.cn, hxyhxy@stu.xjtu.edu.cn, lyj18@mails.tsinghua.edu.cn, and yl6769@nyu.edu.
[1]Code will be published at https://github.com/microsoft/Table2Charts to facilitate future research, once it is approved by an internal review.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467279

ACM Reference Format:
Mengyu Zhou, Qingtao Li, Xinyi He, Yuejiang Li, Yibo Liu, Wei Ji, Shi Han, Yining Chen, Daxin Jiang, and Dongmei Zhang. 2021. Table2Charts: Recommending Charts by Learning Shared Table Representations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467279

1 INTRODUCTION

Creating charts for a multi-dimensional dataset (denoted as table) is a common activity in many domains such as education, research, engineering, finance, *etc.* To discover insights and perform routine analysis, people spend a huge amount of time constructing different types of charts to present diverse perspectives on their tables – such as the charts in Figure 2 created for Table 1a and 1b. Both **data queries** (selecting *what* data to analyze) and **design choices** (*how* to visualize selected data) are made during chart creation [8]. This tedious process requires experience and expertise in data analytics and visualization tools. For example, to compose the bar chart in Figure 2a, one has to first select the left-most three fields/columns from Table 1a, then choose bar chart type, map the three fields onto x and y axis, stack two value series one upon another, *etc.*

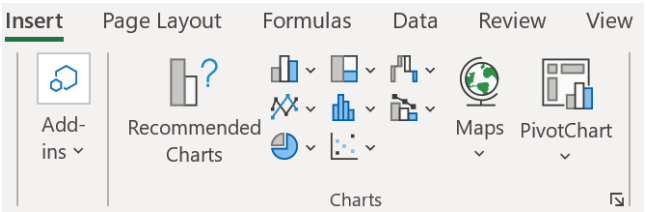


Figure 1: An Example of Chart Creation Entry UI.

To simplify chart composing, a long line of works tried to build machine learning models recommending data queries and/or design choices, such as DeepEye [9], Data2Vis [4], DracoLearn [13] and VizML [8]. However, most of them did not address the **single-type** tasks that each recommends one specific type of charts for a given table, including less used but meaningful minor chart types (*e.g.*, area and radar charts). They only considered the **multi-type** task where a ranked list of few major types of charts (*e.g.*, line, bar, scatter

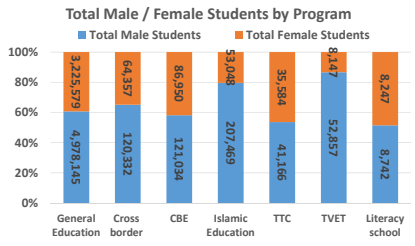
Table 1: Two Example Tables.

(a) Student Statistics Table.

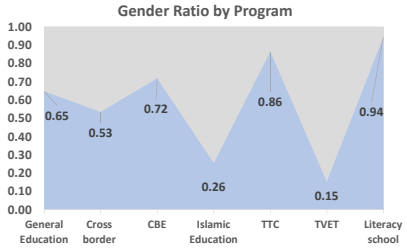
Program	Total Male Students	Total Female Students	Gender Ratio	Total Male Students Percentage	Total Female Students Percentage	Total Students	Total Program Students Percentage
General Education	4,978,145	3,225,579	0.65	60.68%	39.32%	8,203,724	91.03%
Cross border	120,332	64,357	0.53	65.15%	34.85%	184,689	2.05%
CBE	121,034	86,950	0.72	58.19%	41.81%	207,984	2.31%
Islamic Education	207,469	53,048	0.26	79.64%	20.36%	260,517	2.89%
TTC	41,166	35,584	0.86	53.64%	46.36%	76,750	0.85%
TVET	52,857	8,147	0.15	86.65%	13.35%	61,004	0.68%
Literacy school	8,742	8,247	0.94	51.46%	48.54%	16,989	0.19%

(b) Evapotranspiration and Wind Table.

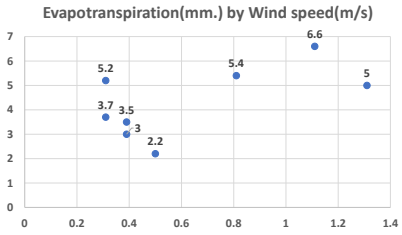
Date	Evapotranspiration (mm.)	Dir.	Wind speed (m/s)
1-May-2010	5.4	N	0.81
19-May-2010	6.6	NE	1.11
6-Jul-2010	3.0	E	0.39
3-Aug-2010	3.5	SE	0.39
1-Sep-2010	5.0	S	1.31
12-Sep-2010	3.7	SW	0.31
23-Sep-2010	2.2	W	0.50
14-Oct-2010	5.2	NW	0.31



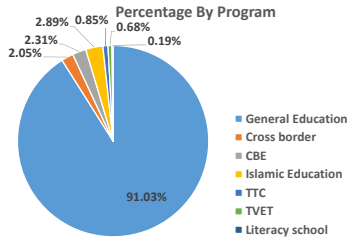
(a) Bar Chart for Table 1a.



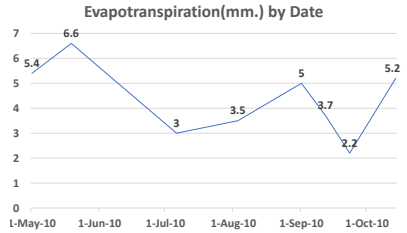
(b) Area Chart for Table 1a.



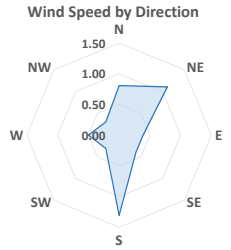
(c) Scatter Chart for Table 1b.



(d) Pie Chart for Table 1a.



(e) Line Chart for Table 1b.



(f) Radar Chart for Table 1b.

Figure 2: Example Charts for Table 1a and Table 1b.

and pie charts) are recommended together. But both single and multi-type tasks should be tackled for real-world scenarios: When facing a table for the first time, one usually has no clear idea about what chart should be created. In this scenario, an assistant could help leverage past common wisdom of what commonly composed charts could be created for the table – which is the *multi-type* task. For example, in Excel, the "Recommended Charts" button in Figure 1 is expected for this. Later with a clearer intention in mind, the main obstacle is the efforts needed to realize ideas through the complex charting process. Since lots of charting tools put chart type buttons / choices as the top entry points to chart composing (*e.g.*, the chart type icons in Figure 1), guessing and suggesting auto-filling and completion of the details of a chosen chart type – which is the *single-type* task – could help save time and efforts from users. For example, when the first three fields of Table 1a are selected, after clicking the bar chart icon on Figure 1, a lot of efforts could be saved if the rest of design choices on field mapping and stacking could be done automatically, leading to the bar chart in Figure 2a.

When tackling the single-type and multi-type tasks with both data queries and design choices, there are three fundamental challenges. First, **separate costs**: It is memory and speed inefficient to design, train and deploy models for multi-type task and single-type tasks repeatedly and independently. Second, **imbalanced data**: The available data for different chart types are highly imbalanced. Four major types of charts (line, bar, scatter and pie) cover 98.91% of the available charts while others rarely appear because it is hard for non-experts to create them. Lack of data in minor chart types (area and radar) makes it hard to build high-quality models for them. Third, **table as context**: Selecting and visualizing data from a table depend on not only the data statistics, but also the semantic meanings of the whole table context. Proper models need to be designed to take table context into chart recommendation.

In this paper, we propose **Table2Charts** framework to learn common patterns of chart creation – including both data queries and design choices – from a large amount of (table, charts) examples and to recommend charts for each given table. In §2, charts recommendation is formulated as table to sequence(s) problem with next-action-token estimation to fill chart template(s). This formulation allows chart recommendation with partial intent when part of data queries and design choices are already given. Then in §3, as the estimation heuristic for beam searching, we design an encoder-decoder deep Q-value network (DQN) which selects table fields to fill template(s) via copying mechanism. All recommendation tasks share one encoder but have their own decoders, which addresses the separate costs challenge. The DQN is trained using mixed learning on the multi-type task of major chart types. By exposing its encoder part to the diverse source tables of different chart types, it learns **shared table representations** containing semantic and statistic information of table fields. Then the pre-trained table representations are transferred for type-specific decoders of single-type tasks, relieving the imbalanced data problem.

From the public web, we collect a large corpus of 266252 charts created from 165214 tables in Excel files and use a public Plotly corpus of 67617 charts from 36888 tables to verify the effectiveness of Table2Charts framework in §4. For each chart type, the recall for top-3 and top-1 recommendations are 59.99% $\sim$ 94.04% and 49.30% $\sim$ 79.72% on the single-type tasks. The multi-type task of recommending major chart types has 61.84% recall at top-3 and 43.84% recall at top-1, which exceed the baseline methods whose maximal recall numbers are 27.14% and 13.17% respectively. Human evaluation is also conducted to validate the precision of the proposed framework on 500 frequently visited web tables from a search engine. Lastly, through T-SNE visualization, we find that the DQN could learn shared table representations during multi-type task training for later transfer learning, thus improving the performance and saving memory occupation of single-type tasks. All these experiments and evaluations justify that Table2Charts could efficiently learn to help composing charts.

In summary, our main contributions are:

- Table2Charts framework is proposed by us to learn human chart composing wisdom. It generates both data queries and design choices in an action sequence for multi-type and single-type chart recommendation tasks with the state of the art performance and efficiency.
- To the best of our knowledge, we conduct the largest scale (165k tables and 266k charts from Excel corpus) training with diverse evaluations (on Excel, Plotly, and web table corpora) of chart recommend systems.
- We show the feasibility of learning shared table representations (encoding table fields into embedding vectors) for enhancing down-stream data analysis tasks.

2 PROBLEM

To build machine learning models that learn patterns from large amounts of (table, charts) pairs, and to generate commonly composed charts for a given table, in this section we formulate single-type and multi-type chart recommendation tasks as table to action sequences generation by filling chart grammar templates.

A **table** here is an n-dimensional dataset $\mathcal{D}$ which contains n data fields $\mathcal{F}_\mathcal{D} = (f_1^\mathcal{D}, \cdots, f_n^\mathcal{D})$. Each data field refers to an attribute of the dataset with its corresponding header name (attribute metadata) and data values (records). For example, each column from tables in Figure 1 is a data field with its first row as header.

To demonstrate our ideas, as shown in Figure 2, in this paper we pick four **major** and two **minor chart types** that appeared in common charting tools such as Excel. The major types are line, bar, scatter and pie charts. The minor types are area[2] and radar[3] charts.

2.1 Chart Templates

Although different types of charts exhibit distinct visual effects and behaviors, the essential actions for creating them from tables can be summarized into two categories: Selecting / referencing table fields and running specific charting commands / operations to organize and plot the selected fields. In this sense, a chart can be regarded as a sequence of actions on data queries and design choices.

Definition 1 (Action Space / Tokens)**.** For an n-dimensional table $\mathcal{D}$, there are two categories of action tokens $\mathcal{A}_\mathcal{D} = \mathcal{F}_\mathcal{D} \cup C$ representing core actions of composing a chart:

- *Field referencing token* $f \in \mathcal{F}_\mathcal{D}$ that indicates a field is selected for composing chart.
- *Command tokens* (denoted as C) which defines other commands for structuring a chart, including:
  (1) Chart type tokens, such as `[Line]` means to start composing a line chart sequence;
  (2) Separator `[SEP]` which splits the referenced fields with different roles in a chart sequence;
  (3) Group operations in $\mathcal{G} = \{$`[Cluster]`, `[Stack]`$\}$[4] indicating how to put multiple data values from multiple fields (series) together along the x axis.

Then we can define how to represent different types of charts using these unified action tokens. Unlike flexible language modelling in NLP, here action tokens should be organized into a sequence according to specific grammar rules of a chart type.

Definition 2 (Chart Grammar Templates)**.** The grammar templates of each chart type can be defined in the Backus-Naur form:

⟨Line⟩ ⊨ `[Line]`⟨f+⟩`[SEP]`⟨f*⟩`[SEP]`
⟨Bar⟩ ⊨ `[Bar]`⟨f+⟩`[SEP]`⟨f*⟩⟨grp⟩
⟨Scatter⟩ ⊨ `[Scatter]`⟨f⟩`[SEP]`⟨f⟩`[SEP]`
⟨Pie⟩ ⊨ `[Pie]`⟨f⟩`[SEP]`⟨f*⟩`[SEP]`
⟨Area⟩ ⊨ `[Area]`⟨f+⟩`[SEP]`⟨f*⟩`[SEP]`
⟨Radar⟩ ⊨ `[Radar]`⟨f+⟩`[SEP]`⟨f*⟩`[SEP]`

where ⟨grp⟩, ⟨f*⟩, ⟨f+⟩ and ⟨f⟩ are token placeholders: ⟨grp⟩ ⊨ an operation $\in \mathcal{G}$, ⟨f*⟩ ⊨ λ | ⟨f⟩⟨f*⟩, ⟨f+⟩ ⊨ ⟨f⟩ | ⟨f⟩⟨f+⟩, and ⟨f⟩ ⊨ *a field* $\in \mathcal{F}_\mathcal{D}$, λ means empty. The first ⟨f+⟩ or ⟨f⟩ segment is

[2] Area Chart is very similar to line chart. The difference is that in area chart, the area between axis and line are commonly emphasized with colors or textures, so that the scale of color fill indicates the volumes. Commonly, area charts are used to represent accumulated totals using numbers or percentages over time.
[3] Radar chart is used to compare the properties of a single component or the properties of two or more variables together.
[4] `[Cluster]` means the values from several fields are put side-by-side, while `[Stack]` means accumulating them one-upon-another for each x category / label (*E.g.*, Figure 2a).

the **y-field**(s) and the second ⟨f*⟩ or ⟨f⟩ segment is the **x-field**(s). [5] Note that how x and y axes behave also depends on chart type: *E.g.*, scatter and pie charts only allow one y-field; temporal records will be ordered by their timestamps along x-axis on a line chart; *etc.*

Hard constraints are also included in the template definitions to restrict heuristic beam searching (see §3). These could be any hand-written rules, such as the data type of a field mapping to y-axis is forbidden to be string type. Currently we only set field type and field number limitations and let Table2Charts models to learn the rest. More rules such as the ones in Draco [13] could be adopted as hard constraints to further improve the framework.

With the above definitions, now each chart can be written down as an action sequence. For example, the sequence of the bar chart in Figure 2a (created from Table 1a) is `[Bar]` (Total Male Students) (Total Female Students) `[SEP]` (Program) `[Stack]`.

There are more detailed charting aesthetics [20] to consider, such as shape, size, color, line width and type, *etc.* In this paper, rather than considering every detail, we mainly focus on the core parts of data queries and design choices – how to select and compose fields as axes of proper chart type – to study if Table2Charts framework can learn common wisdom in a data-driven way. Meanwhile, for the sake of simplicity, we only deal with database-like tables and referencing a whole field without filtering, aggregation, bucketing or ordering. All the above subtle aspects of analysis may not be well supported by our training data (Excel files on the public web, see §4.1) in both quantity (< 5% charts involve customizing them) and quality (the creators of these files may not be experts on inessential parts of charting). They can still easily be added as new tokens into the action space and grammar templates in the future.

2.2 Table to Sequence Generation

Table to charts recommendation now becomes how to meaningfully fill the placeholders of chart template(s). In other words, how to learn common wisdom to generate action token sequences (token-by-token from left to right) that follow the grammars of the given template(s). Note that in a single-type task the first chart type token is fixed (the generation starts from the second token), while in the multi-type task it starts from the first chart type token.

A common way to solve sequence generation problem is to learn an estimation function for heuristic beam searching (more details in appendix §A.3). Given a table $\mathcal{D}$ and an incomplete chart sequence s, we can define the valid actions space of the sequence as $\mathcal{A}_\mathcal{D}(s)$ according to its corresponding template. Follow the language modelling formulation in [24], we choose $Q(s,a) = P(sa \in \mathcal{T}^+_\mathcal{D} \mid s, \mathcal{D})$ as action-value function to guide the choice of next action token $a \in \mathcal{A}_\mathcal{D}(s)$. Here $\mathcal{T}^+_\mathcal{D}$ is the set of all target chart sequences (the charts that would be adopted by user for $\mathcal{D}$) and their prefixes. So the optimal action-value function $q_*(s,a) = \begin{cases} 1 & \text{if } s' = sa \text{ and } s' \in \mathcal{T}^+_\mathcal{D}, \\ 0 & \text{otherwise.} \end{cases}$ is the learning target for $Q(s,a)$. More details of the corresponding Markov decision process can be found in appendix §A.1.

[5] **X-fields** are the fields mapped to x-axis in line, bar, scatter and area charts, to legend in pie charts, and to curved polar axis in radar charts. Multiple x-fields means concatenation. **Y-fields** are the fields mapped to y-axis in line, bar, scatter and area charts, to the size of slice in pie charts, and to radial axis in radar charts. Multiple y-fields means multiple value series are shown together.

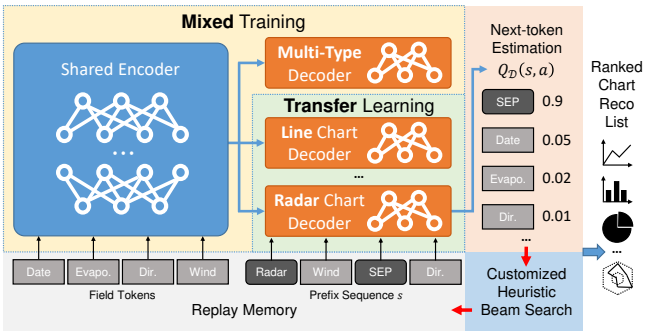


Figure 3: Overview of Table2Charts Framework.

3 METHOD

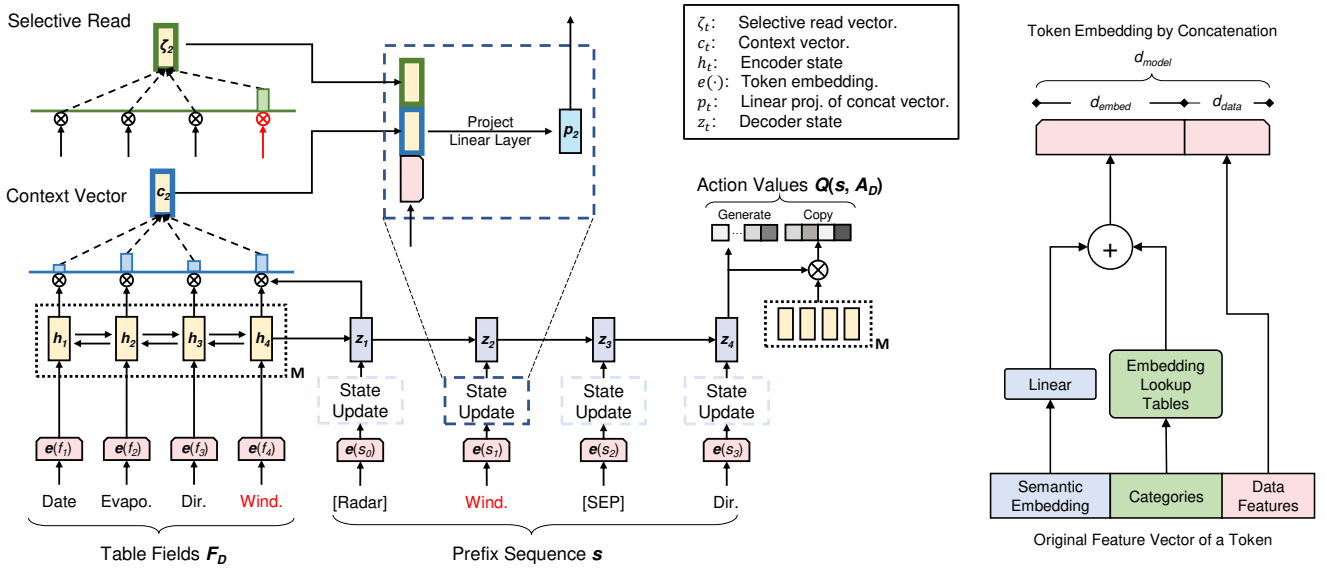
An overview of Table2Charts framework is shown in Figure 3. To approximate $q_*(s,a)$ for chart generation, in §3.1 we design an encoder-decoder deep Q-Network (DQN) architecture with copying mechanism. Because the exposure bias is severe for sequence generation with templates, in §3.2 we adopt search sampling technique to train DQN during beam searching. Finally, in order to solve the imbalanced data problem and mutually enhance the performance among different chart types, in §3.3 we propose a mix-and-transfer training paradigm for all the single and multi-type tasks.

3.1 Filling Templates: DQN with Copying

As shown in Figure 4, we design a DQN (deep Q-network) $Q(s, \mathcal{A}_\mathcal{D})$ to approximate $q_*(s,a)$. $Q(s, \mathcal{A}_\mathcal{D})$ takes all the fields $\mathcal{F}_\mathcal{D} = (f_1, ..., f_n)$ and a state $s = s_0 ... s_{T-1}$ as its input, and calculates the estimated action values ($\in [0,1]$) for all $a \in \mathcal{A}_\mathcal{D}$. Only the outputs for $\mathcal{A}_\mathcal{D}(s)$, *i.e.* the valid actions *w.r.t.* the template grammar of s, are considered.

In Figure 4a is our customized CopyNet architecture. As shown in Figure 4a, the output vector of $Q(s, \mathcal{A}_\mathcal{D})$ consists of two parts: "Generate" (for the command tokens) and "Copy" (for the field tokens). The "Generate" part contains the action value estimations for command tokens $\mathcal{C}$, which comes from a full connected layer with a binary softmax applied on the final decoder state z_T. The "Copy" part has variable length of value estimations for $\mathcal{F}_\mathcal{D}$, which comes from a binary softmax applied on the product of z_T and a non-linear transformation of the memory $M = \{h_1, ..., h_n\}$ (the encoder outputs). We adopt GRU [2] in bidirectional and unidirectional ways for the encoder and decoder RNN parts, respectively. Thus M is simply the outputs of a bidirectional GRU for $\mathcal{F}_\mathcal{D} = (f_1, ..., f_n)$. A decoder state z_t is updated by p_t from the previous state z_{t-1} in GRU cell. p_t is a linear projection from the concatenated vector of three parts: selective read vector ζ_t, context vector c_t and the token embedding $e(s_{t-1})$. Selective read vector[6] choose the field representation from M for a field token: $\zeta_t = \begin{cases} h_\tau & f_\tau = s_{t-1}, \\ \vec{0} & \text{otherwise.} \end{cases}$ Context vector c_t is a linear attention between z_{t-1} and M: $c_t = \sum_{\tau=1}^{n} \frac{e^{\eta(z_{t-1}, h_\tau)}}{\sum_{\tau'} e^{\eta(z_{t-1}, h_{\tau'})}} h_\tau$ where $\eta(\cdot,\cdot)$ is a linear function on two vectors.

[6] Since each field token in s can only refer to one unique field from $\mathcal{F}_\mathcal{D}$ rather than possibly multiple source tokens in the original CopyNet, the calculation of selective read vector ζ is simplified in $Q(s, \mathcal{A}_\mathcal{D})$.



(a) Customized CopyNet Overview. **(b) Token Embedding $e(\cdot)$.**

Figure 4: $q_*(s, a)$ Approximator: DQN Model Architecture.

The token embedding part omitted in Figure 4a is shown in Figure 4b. It is part of encoder and is shared with decoder. Three kinds of token features (details in §A.2) are fused together: 1) Semantic embedding of header name using FastText [1]; 2) Categories including token type, field data type, *etc.*; 3) Data features about the statistics and distribution information of data values.

There are several differences between our model and the original CopyNet architecture [7]: First, unlike the typical NLP scenario where vocabulary size is far greater than the length of the copying source, vocabulary (the command tokens) in Table2Charts is small while the universe of table fields is infinite, and there is no overlapping between generate mode (for vocabulary) and copy mode (for table fields). Second, the input tokens to $Q(s, \mathcal{A}_\mathcal{D})$ first go through the feature transformation network $e(\cdot)$ in Figure 4b rather than the usual index to embedding matrix in NLP. Third, the output of $Q(s, \mathcal{A}_\mathcal{D})$ is a vector of $[0, 1]$ values for each action, rather than a probability distribution over all actions in the original CopyNet.

Our design of DQN with copying mechanism is naturally suited for tasks generating structures from table fields. It handles the open vocabulary of table field universe and provides a clear division between table representation (encoder) and template filling (decoder). The encoder part takes in the whole **table context** and generate field embedding vectors as table representations. The decoder part consumes these vectors for next-token estimation. As good $q_*(s, a)$ estimator, $Q(s, \mathcal{A}_\mathcal{D})$ is then used by Table2Charts as a heuristic function in beam searching to generate multiple sequences.

3.2 Fixing Exposure Bias: Search Sampling

A traditional way to train a next-token estimator is through teacher forcing [21] by only sampling the prefix sequences of user-created charts, and comparing the estimated actions with actual user actions. In other words, in teacher forcing the only samples used to train $Q(s, \mathcal{A}_\mathcal{D})$ network come from a corpus of (table, charts) pairs following the format of $q_*(s, a)$ (see §2.2) with $s \in \mathcal{T}_\mathcal{D}^+$.

As discussed in [14, 24], with only teacher forcing, the outcome model could face exposure bias problem which is common in sequence generation. During teacher forcing, the model is only exposed to the ground truth states (target prefixes); While at inference time it has only access to its own predictions. As a result, during generation it can potentially deviate quite far from the actual sequence to be generated, leading to a biased estimation.

To mitigate exposure bias, we take the search sampling approach in [24] to close the gap between training and inference. Inspired by reinforcement learning, the search sampling process adopts $Q(s, \mathcal{A}_\mathcal{D})$ as the heuristic function to conduct beam searching on each table (details in appendix §A.3). Then the expanded states (including negative samples, s not in $\mathcal{T}_\mathcal{D}^+$) will be stored in a replay memory for periodical update of $Q(s, \mathcal{A}_\mathcal{D})$ itself. This process is very effective after the warm-up of the network with teacher forcing. Without search sampling, the model would perform poorly with the customized beam searching process limited by chart templates.

3.3 Mixed Training and Transfer Learning

As discussed in §1, for single and multi-type tasks, there exists separate costs and imbalanced data challenges. As shown in Figure 3, our basic idea to solve the challenges is that the table representations (the encoder part) can be shared by several (one multi-type and six single-type) tasks. This exposes the encoder to diverse and abundant table field samples, and reduce the memory occupation and inference time for deploying models of the tasks.

To train the shared table representation encoder and the task-specific decoders, as shown in Figure 3, we propose a *mix-and-transfer* paradigm containing two stages: 1) **Mixed Training**: Mixing samples from all major chart types together and train one DQN

model. Its mixed encoder will be transferred to the next stage, while the whole mixed DQN will be used for the multi-type recommendation task. 2) **Transfer Learning**: Take the mixed encoder from the previous stage and freeze its parameters. Then, for each single-type task, a new decoder is trained with the fixed shared encoder using only the data of this chart type.

Comparing to **Separate Training** where a whole DQN is trained for each single-type task (using only the data of that chart type), the mix-and-transfer paradigm in Table2Charts has the following advantages: First, smaller memory occupation and faster inference speed, because now DQN models for all tasks share one same encoder, while separate training still inefficiently holds one for each task. This addresses the **separate cost** challenge. Second, the encoder is exposed to far more samples than each individual chart type can provide. This not only leads to better learning and generalization of the table representation (see §4.5 on how **table context** is represented), but also addresses the **imbalanced data** challenge so that only decoder part (which is small comparing to the larger encoder part) needs tuning for minor chart types.

4 EXPERIMENTS

In §4.1, we first introduce the (table, charts) corpora which are used for training and evaluating Table2Charts and other baseline models. Then in §4.2 and §4.3, the performance of mix-and-transfer paradigm (discussed in §3.3) is evaluated for single and multi-type tasks. Further empirical studies are also discussed in §4.4 and §4.5.

The experiments are run on Linux machines with 24 CPUs, 448 GB memory and 4 NVIDIA Tesla V100 16G-memory GPUs. Each training consists of 30 epochs of teacher forcing on 1 node followed by 5 epochs of search sampling (see §3.2) on 8 nodes. For fair comparisons, all evaluations are done on 1 node with the same configuration. By default, all evaluation metrics reported in this section are averaged over 5 runs for experiments with randomness.

4.1 Chart Corpora

Two corpora – Excel and Plotly – are used for training and evaluation. The Excel corpus is created by us to train and evaluate models, but some baseline models do not provide training scripts. To make fair comparisons, in §4.2 we also evaluate every model on a public Plotly corpus [8] without training or fine-tuning on it.

4.1.1 Excel Corpus. Our chart corpus contains 113390 (42.59%) line, 67600 (25.39%) bar, 64934 (24.39%) scatter, 17436 (6.55%) pie, 1990 (0.75%) area and 902 (0.34%) radar charts. They are extracted using OpenXML [11] from Excel spreadsheet files crawled from the public web. Following data preparation steps are also taken:

1) *Cell Reference Cleansing*. X-fields, y-fields and series names[7] are stored as location references to spreadsheet cells (even in another file), which may lead to inaccessible or invalid tables. Charts with these kinds of cell references are removed from the corpus.

2) *Source Table Restoration*. In spreadsheets, a chart object has no reference to its source table. (Only direct cell references are saved.) To restore the region and structure of its source table, we implement a table detection algorithm [5] according to its cell references. A chart will be dropped if its references are not covered by any detected table, the series names are not in the table header region, or the y-field references are not in the table value region.

3) *Combo Chart Splitting*. In the corpus all combo charts are split into simple charts. Several simple charts (even in different types) can be drawn into one combined plot – *e.g.*, draw a line chart over a bar chart. (Note that simple charts can still have multiple x-fields and y-fields.) In this paper, we focus on simple charts and leave recommendation of combo charts as future work.

4) *Table Deduplication*. To avoid the "data leakage" problem that duplicated tables are allocated into both training and testing sets, tables are grouped according to their schemas[8]. Then within each group, same (table, chart) pairs are merged.

5) *Down Sampling*. After deduplication, the number of tables within each schema is very imbalanced – 0.23% schemas cover 20% of these tables. To mitigate this problem, we randomly sample at most 10 unique tables for each unique (schema, chart) pair.

After preparing the data, 266252 charts are remained in 165214 unique tables with 98588 different schemas. The schemas (with their tables and charts) are randomly allocated for training, validation and testing in the ratio of 7:1:2.

4.1.2 Plotly Corpus. We also adopt the public Plotly community feed corpus [8] and sample 36888 tables with 67617 charts (22644 line charts, 20053 scatter charts, 24204 bar charts and 716 pie charts) from it for testing in §4.2. To extract (table, charts) pairs, following the data processing procedure in VizML [8], we download the full corpus (205GB) and adopt data cleansing code from VizML to remove charts with missing data. Also, similar procedures of *combo chart splitting*, *table deduplication* and *down sampling* are applied to the remaining (table, charts) pairs as in the Excel corpus.

4.2 Evaluations on Multi-Type Reco Task

As mentioned in §3.3, for multi-type task, a mixed DQN is first trained using samples of Excel major chart types. Then, this mixed-trained DQN is used as heuristic function for beam searching to generate a ranked list of major-type charts for each table. We compare Table2Charts framework on recall and precision with four baselines: DeepEye [9], Data2Vis [4], DracoLearn [13] and VizML [8].

4.2.1 Baselines. *DeepEye* (https://github.com/Thanksyy/DeepEye-APIs) provides two public models (ML and rule-based) without training scripts. Thus, we adopt its models without training on our Excel corpus. Because its ML approach works better than its rule-based one on Excel test set, only its ML results are reported in this paper. *Data2Vis* (https://github.com/victordibia/data2vis) model was originally trained on 11 tables with 4.3k charts. For fair comparison, we re-train its model using our larger Excel training set (see §4.1.1) which is also used by Table2Charts. *DracoLearn* (https://github.com/uwdata/draco) provides inference API without training scripts. Again, we evaluate it without training on our Excel chart corpus. It differs from the other methods in that it needs human defined rules as constraints and focuses on searching for specified chart components that least violates them. Thus, in Draco we adopt its default rules and specify it to generate chart type, x-fields and y-fields. *VizML* (https://github.com/mitmedialab/vizml)

[7] **Series names** refer to the name and meaning of each y-field, which are usually displayed in chart legend.

[8] Two tables are defined to have the same **schema** if they have the same number of fields, and each field's data type and header name are correspondingly equal.

Table 2: Evaluations of Table2Charts and Baseline Methods on Multi-Type Reco Task. (Averaged over 5 runs.)

Dataset	Stage	Recall	DeepEye	Data2Vis	VizML	Table2Charts
Excel	Data Queries	R@1	34.33%	31.16%	-	**64.96%**
		R@3	47.13%	42.18%	-	**77.88%**
	Design Choices	R@1	17.83%	26.26%	21.38%	**57.69%**
		R@3	20.35%	48.88%	-	**77.59%**
	Overall	R@1	10.18%	13.17%	-	**43.84%**
		R@3	15.85%	27.14%	-	**61.84%**
Plotly	Data Queries	R@1	49.99%	63.78%	-	**83.34%**
		R@3	62.80%	71.82%	-	**92.02%**
	Design Choices	R@1	37.69%	13.15%	30.21%	**40.17%**
		R@3	37.70%	32.85%	-	**55.57%**
	Overall	R@1	25.05%	16.42%	-	**33.37%**
		R@3	35.98%	33.96%	-	**48.03%**

formulates the design choices into five classification problems and does not recommend data queries. In other words, VizML lacks ability to recommend charts without field selections. Thus, after re-training the classification models on the Excel training set, we only test VizML performance on design choices. More details of experiment setup can be found in appendix §B.

4.2.2 Large-scale Evaluations on Recall. Recall of user charting actions in three stages – data queries, design choices and overall chart recommendation – are evaluated on Excel (testing set) and Plotly (whole dataset) corpora for Table2Charts and the four baselines. On data queries, we examine whether the recommended fields match user-selected ones. On design choices, we evaluate whether models can recommend correct chart type, field mapping and bar grouping operation given the user-selected fields. On overall chart recommendation, both data queries and design choices are compared with the ground truth. Recall at top-k ($k = 1, 3$; R@1, R@3) numbers are adopted as evaluation metrics. They show how a ranked list of chart recommendations matches the user-created charts. More details of recall calculation can be found in appendix §B.2.

The recall numbers are shown in Table 2. We can see that Table2Charts outperforms the baseline methods for all three stages on both Excel and Plotly corpora. The overall R@1 and R@3 have reached 43.84% and 61.84% on Excel (33.37% and 48.03% on Plotly), which exceeds those of baselines by large margins (at least doubled on Excel). The recall numbers of data queries stage are higher on Plotly than Excel – This is because in Plotly corpus, each table only contain fields which are used in corresponding chart, and thus lower the difficulty of selecting fields. In addition to the results in Table 2, DracoLearn has R@1 < 1% on all stages – It needs human-defined rules as constraints and focuses on searching for charts that least violates them, which leads to weak generalization.

4.2.3 Human Evaluation on Precision. To evaluate the quality of recommended charts (precision[9]), we build a labelling website to collect and compare ratings for the top-1 recommendations from Table2Charts, DeepEye and Data2Vis[10]. 500 unique HTML tables crawled from the public web are selected based on query-frequency in a search engine. 10 experts working on web-table visualization manually label in the following way: For a given table, the website shows the table content for reading. When an expert confirms understanding the table, 3 charts recommended by the 3 models will be rendered with the same visualization library and shown in random order anonymously. Three 1 to 5 integer ratings (higher score indicates better chart) are then labelled by the expert. Additionally, the expert is asked to mark if the table is actually unsuitable for chart recommendation. For every (table, 3 recommendation) pair, we collected results from 3 experts to avoid labelling bias.

Table 3: Summary of Human Evaluation Ratings

Rating	5	4	3	2	1	Avg	≥4	≥3	≤2
Table2Charts	517	158	115	102	98	3.90	675	790	200
Data2Vis	309	178	167	125	211	3.25	487	654	336
DeepEye	312	166	139	137	236	3.18	478	617	373

We filtered out the tables which marked as "unsuitable for chart recommendation"[11], and got the distribution of the ratings based on 330 tables left. As shown in Table 3, Table2Charts has the highest average score, the largest amount of good charts (rating=5, rating≥4, rating≥3), and the smallest amount of bad charts (rating ≤2).

To check statistical significance, we further conduct Wilcoxon signed-rank test [19] which is a non-parametric statistical hypothesis test used to compare two related or matched samples to assess whether their population mean ranks differ (*i.e.* it is a paired difference test). At 95% confidence level, when comparing Table2Charts with DeepEye and comparing Table2Charts with Data2Vis, both p-values from Wilcoxon test are less than 0.0001. These results show that the recommended charts from Table2Charts have better quality than those from DeepEye and Data2Vis.

4.2.4 Efficiency Comparison. On average, Table2Charts only takes 12.14ms to generate chart recommendations for a table, while it costs DeepEye and Data2Vis 48.19ms and 210ms, respectively. In summary, Table2Charts outperforms baseline methods on both performance and efficiency.

4.3 Evaluations on Single-Type Reco Tasks

After mixed training, as discussed in §3.3, the shared table representation encoder is taken and frozen for the training of six decoders for six single-type tasks. For comparison, the separate training (see §3.3) will generate an independent DQN model for each chart type with the same settings as the transfer learning. Also, the mixed DQN from §4.2 is directly tested on single-type tasks of major types.

The evaluation results are shown in Table 4. The mix-and-transfer paradigm ("Transfer") has higher recall numbers than separate training ("Separate") and mixed-only ("Mixed") DQN for all chart types. Table2Charts could handle single-type tasks well by learning shared **table context** representations. Considering the model size where the encoder and decoder parts are designed with 1.3M and 0.5M parameters (see appendix §B.1.1), "Separate" have 10.8M parameters while "Transfer" only has 4.3M parameters. In this way, Table2Charts reduces **separate costs** and improves efficiency for model deployment and inference.

[9]Precision numbers cannot be calculated from Excel and Plotly corpora because they only have user-created charts but there can be good charts not created by users.
[10]VizML and DracoLearn are not included because VizML cannot recommend complete chart with data queries and DracoLearn has weak generative power.
[11]Determining whether a table is suitable for generating a chart is out of the scope of this paper and would be part of future work.

Table 4: Evaluations of Three Training Methods (§3.3) on Six Single-Type Tasks. (Averaged over 5 runs.)

Type(s)		Recall	Separate	Mixed	Transfer
Major Type	Line	R@1	52.02%	52.53%	**53.78%**
		R@3	68.28%	68.58%	**69.37%**
	Bar	R@1	56.56%	58.69%	**60.25%**
		R@3	70.34%	72.07%	**73.14%**
	Scatter	R@1	51.73%	54.69%	**56.48%**
		R@3	69.33%	68.96%	**74.24%**
	Pie	R@1	73.60%	77.99%	**79.72%**
		R@3	90.60%	93.12%	**94.04%**
Minor Type	Area	R@1	27.48%	–	**49.30%**
		R@3	40.32%	–	**59.99%**
	Radar	R@1	49.90%	–	**71.77%**
		R@3	60.93%	–	**77.00%**

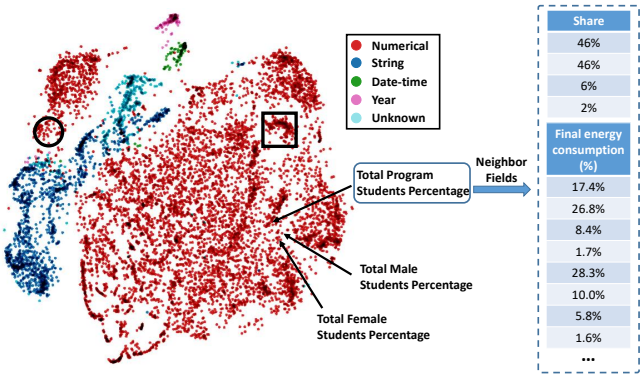


Figure 5: Visualization of Shared Table Representations.

As for minor chart types, there are huge gaps between the performance of "Transfer" and "Separate" in Table 4. With mix-and-transfer ("Transfer") paradigm, on average R@1 and R@3 increase 21.84% and 17.87%. The main reasons are that the shared encoder could capture and extract the information of table context and field semantics, and the quantity of minor type charts are sufficient to train decoder which is of smaller size. Thus, as discussed in §3.3, the **imbalanced data** problem is overcome.

4.4 Recommendation Case Studies

As an example, in this section we take Table 1a (with user-created Figure 2a and 2b) to conduct empirical studies. When a user does not know where to start, the multi-type mixed model (§4.2) recommends common types of charts. Its top-3 recommendations are:

1) [Bar](Total Male Students)(Total Female Students)[SEP](Program)[Stack]
2) [Bar](Total Male Students)(Total Female Students)[SEP](Program)[Cluster]
3) [Bar](Total Program Students Percentage)[SEP](Program)[Cluster]

From the above results, we can see that the mixed model successfully recommends the bar chart in Figure 2a as the top-1 result. Our model can identify f_2^{1a}-f_3^{1a} as a group and use them to create a bar chart with two y-fields. Result 2) is the clustered form of the bar chart in result 1). Result 3) is also useful. Our model can identify f_8^{1a} should not be a measure in the f_2^{1a}-f_3^{1a} group. The multi-type model tend to recommend what are commonly composed, thus may lack diversity (*e.g.*, all the above results are bar charts). So one can also put single-type recommendations into the list. (We leave as a future work how to mix the results from multi-type and single-type models together for a balanced recommendation.)

When a user has chosen a specific chart type and needs auto-completion help, it is time to use single-type models (§4.3) for recommendations. In such scenario, our single-type models can recommend all as top-1 the area chart in Figure 2b and the bar chart in Figure 2a. Furthermore, the single-typed model can also recommend the pie chart in Figure 2d, which is also meaningful to show the percentages but not originally created by the user.

4.5 Exploring Table Representations

To understand how the embeddings generated by the shared table representation encoder work, from the validation set 20000 fields (from 3039 tables) are randomly chosen and visualized through t-SNE [16]. In the left part of Figure 5, each point represents a field and the color represents its field type. In the figure, we can see the field type information is learnt by the embedding in a meaningful way. For example, date-time fields and year fields are close. One possible explanation is that they both are often used as x-axes in line charts, and thus have similar representations.

As depicted in Figure 5, marked by arrows are the points corresponding to the fields f_5^{1a}, f_6^{1a} and f_8^{1a} (which is "Total Program Students Percentage") from Table 1a. They are close to each other because their record values are all percentages. Note that f_5^{1a} and f_6^{1a} are closer compared to f_8^{1a} because their semantics are similar (contain gender information). Some example neighbor fields (based on cosine distance) of f_8^{1a} are shown in the right part of Figure 5. Similar to f_8^{1a}, these fields are also percentages that sum up to 1.

Two more clusters are shown as examples in Figure 5. In the squared area, there are many fields about countries. *E.g.*, there are four numerical fields (from four tables) with header names "U.S.", "Japan", "England" and "Scotland" showing annual statistics. In the circular area, many fields take the role of index or ID. *E.g.*, located in this cluster are four fields (from four tables) with header names "VerminID", "Index", "category" and "Course Code" and increasing integers. These integers lose the measure property – they are not for mathematical operations / aggregations. Thus, these fields locate very close to string fields (dark blue points) in Figure 5.

5 RELATED WORK

Analysis Recommendation: For general data analysis and insight recommendation from tables, current systems are mostly based on collaborative filtering [10], statistical significance [17], heuristic and history matching [6, 12], or only target for specific analysis [24]. They rarely consider the semantic meaning of table context or tackle the challenges in recommending multiple types of analysis, which are both taken into account by Table2Charts in an end-to-end approach using large-scale human created corpus.

Chart Recommendation is an important branch in analysis recommendation. Lots of visualization recommendation systems heavily rely on hand-crafted heuristics and rules, such as Voyager [22] and DracoLearn [13]. Data-driven approaches are becoming popular in recent learning-based systems such as DeepEye [9], Data2Vis [4] and VizML [8]. In §4.2, we discussed DeepEye,

Table 5: Comparisons among Different Chart Recommending Systems

System	Reco Tasks	Learning Approach	Models	Dataset	N_{data}	Data Source	Data Generation
Table2Charts	Data queries + Design choices	End-to-end chart generation as action token sequence	CopyNet as deep Q-network	(full table, charts) pairs	165k tables with 266k charts	Web Excel files	Human
VizML	Design choices	5 classification tasks	Fully-connected feed-forward NN	(partial table, charts) pairs	119k tables	Plotly community feed	Human
DracoLearn	Data queries + Design choices	Soft constraints/rules weights for clingo ASP solver	RankSVM	Pairwise comparison	1100 + 10 pairs	Various	Rules → Annotations
Data2Vis	Data queries + Design choices	End-to-end chart generation as JSON string sequence	Character-level seq2seq NN	(full table, charts) pairs	11 tables with 4300 charts	Tool (Voyager)	Rules → Validations
DeepEye	Data queries + Design choices	1. Good/bad classification 2. Ranking	1. Decision tree 2. LambdaMART	1. Good/bad chart labels 2. Pairwise comparison	42 tables with 1. 33.4k labels 2. 285k pairs	Various	Rules → Annotations

Data2Vis, DracoLearn and VizML as baselines. More of their differences with Table2Charts are summarized in Table 5. DracoLearn and DeepEye both learnt from low quality data and depended on complex rule designs. Data2Vis suffered from naive model of character-level seq2seq generation of Vega-lite [15] JSON string. VizML only considered design choices without handling data queries.

Structured Prediction: Filling chart templates and generating action sequences is a structured prediction problem. There are lots of related work such as NL QA and Text2SQL [18]. Table2Charts is inspired by [7, 24] to design an encoder-decoder architecture with copying mechanism as a function approximator.

Representation Learning and Pre-training: The word embedding [1] and pre-training paradigm [3] in NLP inspired us to learn pre-trained table representations for multiple tasks [23]. The table field embedding could be useful for more down-stream data analysis tasks including recommendation of other types of analysis.

6 CONCLUSION

In this paper, we propose the Table2Charts framework to solve single and multi-type chart recommendation tasks considering both data queries and design choices. Through copying from table fields, shared table representations are learnt to enhance performance and efficiency for all chart types. We believe the proposed techniques can be widely used for data analysis tasks on tables in the future.

REFERENCES

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics (TACL)* 5 (2017), 135–146.

[2] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 1724–1734.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. ACL, 4171–4186.

[4] Victor Dibia and Çagatay Demiralp. 2019. Data2Vis: Automatic Generation of Data Visualizations Using Sequence-to-Sequence Recurrent Neural Networks. *IEEE Computer Graphics and Applications (CG&A)* 39, 5 (2019), 33–46.

[5] Haoyu Dong, Shijie Liu, Zhouyu Fu, Shi Han, and Dongmei Zhang. 2019. Semantic Structure Extraction for Spreadsheet Tables with a Multi-task Learning Architecture. In *Workshop on Document Intelligence at NeurIPS 2019*.

[6] Humaira Ehsan, Mohamed A. Sharaf, and Panos K. Chrysanthis. 2016. MuVE: Efficient Multi-Objective View Recommendation for Visual Data Exploration. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 731–742.

[7] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating Copying Mechanism in Sequence-to-Sequence Learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL, Volume 1: Long Papers)*. ACL, 1631–1640.

[8] Kevin Zeng Hu, Michiel A. Bakker, Stephen Li, Tim Kraska, and César A. Hidalgo. 2019. VizML: A Machine Learning Approach to Visualization Recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 1–12.

[9] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. DeepEye: Towards Automatic Data Visualization. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 101–112.

[10] Patrick Marcel and Elsa Negre. 2011. A survey of query recommendation techniques for data warehouse exploration. In *EDA*. Hermann, 119–134.

[11] Microsoft. 2018. Open XML SDK. https://github.com/OfficeDev/Open-XML-SDK

[12] Tova Milo and Amit Somech. 2018. Next-Step Suggestions for Modern Interactive Data Analysis Platforms. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 576–585.

[13] Dominik Moritz, Chenglong Wang, Greg L. Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. 2019. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 25, 1 (2019), 438–448.

[14] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence Level Training with Recurrent Neural Networks. arXiv:1511.06732 https://arxiv.org/abs/1511.06732

[15] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE transactions on visualization and computer graphics (TVCG)* 23, 1 (2017), 341–350.

[16] Daniel Smilkov, Nikhil Thorat, Charles Nicholson, Emily Reif, Fernanda B. Viégas, and Martin Wattenberg. 2016. Embedding Projector: Interactive Visualization and Interpretation of Embeddings. (2016). arXiv:1611.05469 http://arxiv.org/abs/1611.05469

[17] Bo Tang, Shi Han, Man Lung Yiu, Rui Ding, and Dongmei Zhang. 2017. Extracting Top-K Insights from Multi-dimensional Data. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*. ACM, 1509–1524.

[18] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL, 7567–7578.

[19] Frank Wilcoxon, SK Katti, and Roberta A Wilcox. 1963. *Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test*. American Cyanamid Company Pearl River, NY.

[20] Claus O Wilke. 2019. *Fundamentals of Data Visualization: a Primer on Making Informative and Compelling Figures*. O'Reilly Media.

[21] Ronald J Williams and David Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural computation* 1, 2 (1989), 270–280.

[22] Kanit Wongsuphasawat, Dominik Moritz, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2016. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 22, 1 (2016), 649–658.

[23] Yu Zhang and Qiang Yang. 2018. A Survey on Multi-Task Learning. arXiv:1707.08114 http://arxiv.org/abs/1707.08114

[24] Mengyu Zhou, Wang Tao, Pengxin Ji, Han Shi, and Dongmei Zhang. 2020. Table2Analysis: Modeling and Recommendation of Common Analysis Patterns for Multi-Dimensional Data. In *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, 320–328.

A TABLE2CHARTS FRAMEWORK DETAILS

In this section, we will dive into the details of the table-to-sequence problem formulation in §2.2 with the corresponding Markov decision process. Also, the input token features to the input embedding network of Figure 4b in §3.1 will be listed for your references. Finally, we will describe the companion heuristic beam searching algorithm to the DQN. Core code of Table2Charts and part of the test data can be found at https://github.com/microsoft/Table2Charts.

A.1 Markov Decision Process (MDP)

As described in §2.2, the MDP for chart generation is based on the one for pivot table that was first defined by [24].

Definition 3 (Chart Generation MDP)**.** For a table $\mathcal{D}$, we adopt the definitions in §2.1 and §2.2 to describe the next-token chart sequence generation MDP:

- State space is $\mathcal{S}^+_{\mathcal{D}} = \{s \mid s \in \bigcup_{l=1}^{\infty} {\mathcal{A}_{\mathcal{D}}}^l, s \text{ is legal}\}$, which can be viewed as a forest with chart type tokens (see §2.1) as root nodes (initial states) of the trees. $\mathcal{S}^+_{\mathcal{D}}$ contains all the prefixes of all the possible legal chart sequences that follow the chart templates.
- Action space $\mathcal{A}_{\mathcal{D}}$: The legal actions for a given state s are $\mathcal{A}_{\mathcal{D}}(s) = \{a \mid sa \in \mathcal{S}^+_{\mathcal{D}}, \forall a \in \mathcal{A}_{\mathcal{D}}\}$.
- State transition is deterministic. The transition probability from s to s' by taking action $a \in \mathcal{A}_{\mathcal{D}}(s)$ is:

$$P_{\mathcal{D}}(s, a, s') = \begin{cases} 1 & \text{if } s' = sa, \\ 0 & \text{otherwise.} \end{cases}$$

- Reward function $R_{\mathcal{D}}$ is designed to reflect if a user-created sequence is successfully generated:

$$R_{\mathcal{D}}(s, a, s') = \begin{cases} 1 & \text{if } s' = sa \text{ and } s' \in \mathcal{G}_{\mathcal{D}}, \\ 0 & \text{otherwise.} \end{cases}$$

Here $\mathcal{G}_{\mathcal{D}}$ is a subset of $\mathcal{S}^+_{\mathcal{D}}$ which contains exactly the charts created by user for $\mathcal{D}$. (As first mentioned in §2.2, $\mathcal{T}^+_{\mathcal{D}}$ is the set of all the prefixes of all the target sequences in $\mathcal{G}_{\mathcal{D}}$.)

- Discount rate $\gamma = 1$ so that the length of a chart sequence has no impact on its rewards.

According to Bellman optimality equation, one can easily find the optimal action-value function (the expected discounted return for the optimal policy): $q_*(s, a) = R_{\mathcal{D}}(s, a, sa) + \gamma \max_{a' \in \mathcal{A}_{\mathcal{D}}(sa)} q_*(sa, a')$ $= \begin{cases} 1 & \text{if } s' = sa \text{ and } s' \in \mathcal{T}^+_{\mathcal{D}}, \\ 0 & \text{otherwise.} \end{cases}$ In other words, $q_*(s, a)$ equals to 1 if and only if sa is a prefix of a target sequence. As described in §2.2, the rest of the problem is to learn a good approximator for $q_*(s, a)$.

A.2 Token Features for Input Embedding

As shown in Figure 4b, token embedding vector consists of:

Semantic Embedding. Semantic embedding features are calculated from the header name of a field (*e.g.*, table header or database attribute string). In this work, we adopt FastText [1] with $vocabsize = 200,000$ and $embedsize = 50$ for semantic embedding. If there are more than 1 words in the field name, the embedding of all words are averaged.

Field Categories. There are five types of categorical features which are adopted in this work.

(1) *Token type* shows the type of a token in an analysis sequence, which includes {`PADDING`, `SEP`, `FIELD`, `GRP`, `Line`, `Bar`, `Scatter`, `Pie`, `Area`, `Radar`}.
(2) *Segment type* shows to which segment a token belongs in an analysis sequence. This categorical feature can be {`PADDING`, `X`, `Y`, `GRP`, `OP`}. `OP` corresponds to `SEP` and chart type tokens.
(3) *Field type* shows the type of a field, which includes {`Unknown`, `String`, `Year`, `DateTime`, `Decimal`}.
(4) *Field role* shows whether a field could be one of the left headers of a cross table (detected during source table restoration). The options include {`Invalid`, `Header`, `Value`}.
(5) *Grouping operation* corresponds to ⟨grp⟩, which includes {`Invalid`, `Cluster`, `Stack`}.

Data features. We adopt the 16 statistic features in [24], and design 15 new features: SumIsIn01, SumIsIn0100, Range, Variance, Covariance, AbsoluteCardinality, MedianLength, LengthStdDev, AvgLogLength, ArithmeticProgressionConfidence, GeometricProgressionConfidence, Skewness, Kurtosis, GiniCoefficient, NRows.

All features except AvgLogLength are calculated for numerical fields. All applicable features are calculated for string fields. Most data features are ranged in $[0, 1]$, and for those whose range may be very large, we normalized them by their 99th percentile numbers in the Excel chart corpus. Data statistic features for non-field tokens remain empty (their values are assigned as zeros).

A.3 Heuristic Beam Searching

In search sampling training and beam searching inference stages, we adopt and customize a drill-down beam searching algorithm [24]. It takes the following steps to generate chart sequences:

(1) Initially, the searching frontier only contains the sequence(s) that each consists of one specified chart type token from {`[Line]`, `[Bar]`, `[Scatter]`, `[Pie]`, `[Area]`, `[Radar]`}. Chart types are chosen according to the training or inference task[12].
(2) For each round, the top-*BeamSize* scored partial / incomplete sequences in the frontier will be popped and extended as described below.
   (a) For each state in the beam, greedily drill down (choose a with the highest $Q(s, a)$ to append) until a complete sequence is generated. The complete sequence is put into the result ranking list with $Q(s, a)$ as its score. Each non-optimal state sa ($a \in \mathcal{A}_{\mathcal{D}}(s)$) from each expansion (one calculation of $Q(s, \mathcal{A}_{\mathcal{D}})$) is put into the frontier also with $Q(s, a)$ as its score.
   (b) No more rounds and stop searching if the number of expansions exceeds *ExpandLimit*.

As mentioned in §2.1, to restrict heuristic beam searching and eliminate some nonsense recommendations, hard constraints are defined in chart templates. For example, the data type of a y-field is forbidden to be string type. During training and inference, these hard constraints are also applied by removing illegal actions from $\mathcal{A}_{\mathcal{D}}(s)$ for each expansion in the heuristic beam searching.

This also allows users to specify more constraints on searching. For example, a user could select interested fields from a table and the beam searching can use exactly these fields to suggest charts.

[12] In separate training, transfer learning and single-type inference, only one chart type is used; while in mixed training and multi-type inference, all major types are used.

Table 6: Hyper-parameters of CopyNet Models Sizes.

Model Size	Layers		Input Dim		Hidden Dim		Total Parameters
	Encoder	Decoder	Encoder	Decoder	Encoder	Decoder	
Small	2	1	192	192	128	128	~0.8M
Medium	2	1	320	256	192	192	~1.8M
Large	4	1	384	512	224	256	~4.9M

B TRAINING AND EVALUATION DETAILS

In this section we elaborates detailed setups of experiments in §4 for Table2Charts and other baseline methods, including DeepEye [9], Data2Vis [4], DracoLearn [13] and VizML [8].

B.1 Training Details

B.1.1 Training Table2Charts. The training process of Table2Charts consists of 30 epochs of teacher forcing followed by 5 epochs of search sampling. First, hyper-parameters of the DQN model are selected by conducting a series of preliminary experiments. For semantic embedding (shown in Figure 4b), two pre-trained NLP embedding models are considered: FastText [1] with embedding size of 50 and vocabulary size of 200000, and BERT [3] with embedding size of 768 and vocabulary (subwords) size of 30522. Besides, we consider three different DQN sizes, with different hidden state dimensions and different number of encoder layers (see Table 6).

To choose embedding model and DQN size, we compare their six possible combinations after the teacher forcing training stage on multi-type task. Results show that compared to FastText, BERT increases R@1 for about 2% (from 13.07% to 14.97%), but doubles the training time. Similarly, R@1 of "small", "medium" and "large" models are 13.07%, 13.30% and 15.37% respectively. The "large" model gains about 2% in recalls while the number of parameters is 2.5× "medium" or 6.1× "small" model. To make a trade-off between performance and training costs, we use the FastText embedding and the "medium" model size in all experiments in §4.

The beam searching hyper-parameters ($BeamSize$, $ExpandLimit$) are fixed to (4, 100). For neural network tuning, we use Adam optimizer with ($learning\_rate$, β_1, β_2, ϵ, $weight\_decay$) set to ($1e4$, 0.9, 0.999, $1e-8$, 0.01). Due to GPU memory limitations, for all experiments the batch size is set to 512. During the back propagation at each step, gradients from each process are averaged.

B.1.2 Training Data2Vis. To train and evaluate Data2Vis, we transform Excel and Plotly data to JSON strings. Following data preparation code from Data2Vis, table input is a JSON dictionary containing key-value pairs of field keys to one randomly sampled row, and chart output is a JSON dictionary in a simplified Vega-lite format. For each (table, chart) pair, two samples are generated by sampling two rows from the table. In total, there are 180383 training samples.

Data2Vis uses a character-level seq2seq model with strings as input and output. We set its encoder to be a 1-layer bidirectional LSTM with hidden dimension 256, and decoder to be 2-layer LSTM with hidden dimension 128. These choices make sure the size of the model (~1.94M) is comparable to the that of Table2Charts (~1.8M).

Following the training configurations of Data2Vis, Adam optimizer is used again. According to our Excel corpus, 98% of the source table JSON strings have fewer than 471 characters, while 99% target chart JSON strings have fewer than 130 characters. Therefore, the maximum source length and target length are set to 500 and 130. The vocabulary sizes of source and target are 98 and 42. The model is trained for nearly 30000 steps, with a batch size of 16.

B.1.3 Training VizML. As mentioned in §4.2.1, VizML focuses on design choices and does not provide models for data queries. We re-train VizML models on its *Mark Type* task (corresponding to chart type) and *Is on X-axis or Y-axis* task (corresponding to field mapping), and change its *Mark Type* task from 2, 3, and 6 classification to 4 classification (including line, scatter, bar, pie chart). These models make predictions for one field at a time. So only the labelled fields (selected in user-created Excel charts) are kept for training. Field feature extraction process and model hyper-parameters are identical with the original VizML paper and source code.

B.2 Evaluation Details

B.2.1 Calculating Recall at Top-k. In this section, we elaborate how recall numbers are calculated on data queries, design choices and overall chart recommendation tasks in §4.2 and §4.3.

On data queries, field selection is compared with ground truth. Given a table, if the chosen field set of any top-k recommended chart matches that of any user-created chart of the table, this table is considered to be successfully recalled *w.r.t.* data queries. Thus, recall at top-k of data queries is calculated as $R@k = \frac{\#(\text{Tables successfully recalled})}{\#(\text{Tables})}$.

On design choices, we consider chart type, field mapping (*i.e.*, map the selected fields onto x-axis and y-axis of a chart), and grouping operation (*i.e.*, whether stacked or clustered, only for bar chart). Given a table and one set of its fields, if any top-k recommended chart adopts the field set and matches a user-created chart of the table, then the field set is considered to be successfully recalled *w.r.t.* design choices. Thus, the recall on design choices is calculated as $R@k = \frac{\#(\text{User-created field sets successfully recalled})}{\#(\text{User-created field sets})}$.

On overall evaluation, both data queries and design choices are considered. The equation for overall evaluation is the same as that of data queries, while the "table successfully recalled" here means that any top-k recommended chart complete matches any user-created chart of the table.

B.2.2 Comparing with Baselines. For fair comparisons among chart reco systems, more evaluation details (in addition to those described in §4.2.1) need to be properly handled. In **DeepEye**, bar grouping operations are not considered, and several data transformations (*e.g.*, dimension breakdown and measure binning) are recommended. To make sure the recommended charts from DeepEye matches the definitions in our corpora, during evaluation we ignore the grouping operations in ground truth and drop the charts with breakdown and binning operations. In **Data2Vis**, searching beam size is set to 15 for data query and overall chart recommendation, and set to 30 for design choices. In **DracoLearn**, same as DeepEye, its weights for soft constraints are not re-trained using our Excel training set. In **VizML**, we only evaluate it on design choices. Given a field set, if *Mark Type* and *Is on X-axis or Y-axis* predictions of all fields in the set match any of user-created chart, then the field set is considered to be successfully recalled *w.r.t.* design choices. Only R@1 is calculated for VizML because only one result is available. In **Table2Charts**, $BeamSize$ and $ExpandLimit$ are the same as in search sampling training (see §B.1.1).