Visualization Recommendation with Prompt-based Reprogramming of Large Language Models

Xinhang Li\(^1\)\(^2\)*, Jingbo Zhou\(^2\)†, Wei Chen\(^1\), Derong Xu\(^1\), Tong Xu\(^1\)†, Enhong Chen\(^1\)†

\(^1\)University of Science and Technology of China & State Key Laboratory of Cognitive Intelligence, \(^2\)Business Intelligence Lab, Baidu Research

xinhangli@mail.ustc.edu.cn, zhoujingbo@baidu.com, 
{chenweicw, derongxu}@mail.ustc.edu.cn, {tongxu,cheneh}@ustc.edu.cn

Abstract

Visualization recommendations, which aim to automatically match proper visual charts for specific data tables, can significantly simplify the data analysis process. Traditional approaches in this domain have primarily relied on rule-based or machine learning-based methodologies. These methods often demand extensive manual maintenance and yet fail to fully comprehend the tabular data, leading to unsatisfactory performance. Recently, Large Language Models (LLMs) have emerged as powerful tools, exhibiting strong reasoning capabilities. This advancement suggests their substantial promise in addressing visualization recommendation challenges. However, effectively harnessing LLMs to discern and rationalize patterns in tabular data, and consequently deduce the essential information for chart generation, remains an unresolved challenge. To this end, we introduce a novel Hierarchical Table Prompt-based reprogramming framework, named HTP. This framework aims to integrate multi-dimensional tabular data into LLMs through a strategically crafted prompt learning method while keeping the LLMs’ backbone and weights unaltered. The HTP framework uniquely incorporates a four-level prompt structure, encompassing general, instance, cluster, and column levels. This multi-level approach is engineered to provide a comprehensive understanding of both general distribution and multifaceted fine-grained features of tabular data, before inputting the tabular data into the frozen LLM. Our empirical studies confirm that the HTP framework achieves state-of-the-art performance, marking an advancement in the field of data visualization and analysis.

1 Introduction

Data visualization, which facilitates effective decision-making via better responding the human sensitivity and efficiency in processing visual information (Kosmyna et al., 2018), has become increasingly crucial nowadays. Traditional visualization tools necessitate extensive effort for manual specification, which often requires domain expertise in data analysis. Therefore, large efforts have been made to develop automatic tools for solving the visualization recommendation task, i.e., recommending the proper visual charts.

Generally, prior arts can be divided into two categories. The first, known as Rule-based approaches (Mackinlay, 1986; Mackinlay et al., 2007), relies on expert-designed rules to generate visualizations. However, these approaches not only suffer the massive manual labor for rule maintenance, but also struggle to address the combinatorial explosion problem caused by increasing data dimensions. In contrast, the Machine Learning-based approaches (Dibia and Demiralp, 2019; Li et al., 2021) are investigated to automatically learn the best matching between tabular data and visualization elements. Unfortunately, few of them could fully extract the multi-level features within tables, which severely limits the performance.

Last year has witnessed the prosperity of Large Language Models (LLMs), whose advanced capabilities of LLM in processing and interpreting...
complex data unlocks new possibilities in the field of data visualization. However, significant challenges persist in effectively harnessing the potential of an LLM for visualization recommendation. Firstly, LLMs are typically designed to process sequences of discrete tokens in an unstructured format, whereas tables used for visualization often display highly structured and predominantly numerical characteristics, which leads to a mismatch. Second, the inherent pre-training of LLMs does not naturally encompass the comprehension abilities required to interpret complex tabular data. Third, rather than fine-tuning the entire model, it is often preferable to keep the LLM backbone “frozen”. This approach ensures the model’s versatility in supporting a variety of tasks without compromising its fundamental capabilities.

To this end, we propose a novel Hierarchical Table Prompt-based reprogramming framework (HTP), to adapt LLM for visualization recommendation without altering the backbone structure. The essence of HTP lies in leveraging data-driven multi-level prompts to adaptively reprogram the tabular data across various dimensions, thereby bridging the gap between the structured nature of tabular data, as well as the inherent capabilities of LLMs for textual processing. Specifically, HTP utilizes four levels of prompts as follows:

- **General-level prompt**, which is employed to describe the overall distribution of table dataset, while facilitating information sharing and integration among prompts at various levels, thereby enhancing the generalization performance of LLM.
- **Instance-level prompt**, which connects the individual table instances with various charts, by leveraging the specific distribution of tabular data and corresponding chart.
- **Cluster-level prompt**, which is generated via feature extraction and clustering, and targets enhancing the LLM to capture the implicit patterns that exist within the table datasets, as well as the correlations among patterns.
- **Column-level prompt**, which originates from the inherent structural information of tables, and highlights the columnar organization to improve the column-level information processing, and support cross-column comprehension.

Based on these four prompts, as illustrated in Figure 1, we concatenate prompts at general, instance, and cluster levels with a serialized table before feeding them into LLMs, thereby facilitating table-specific knowledge extraction and integration. Meanwhile, the column-level prompts are prepended to encoded inputs to retain the structural information of tables, thereby improving the distinction among columns, and fostering a contextual understanding between them. The contribution of this paper could be summarized as follows:

- To the best of our knowledge, we are the first to utilize LLMs to explore the visualization recommendation task through soft prompt-based reprogramming, without altering the pre-trained backbone model.
- A novel prompt-based framework is proposed to reprogram the hierarchical table information into multi-prompts, which enhances the comprehension capabilities of LLMs.
- Extensive experiments on real-world datasets demonstrate the effectiveness of the proposed HTP compared with state-of-the-art approaches.

2 Related Work

2.1 Visualization Recommendation

Prior researches on automated visualization recommendation includes rule-based and machine learning-based approaches. Rule-based approaches (Mackinlay, 1986; Roth et al., 1994; Perry et al., 2013; Wongsuphasawat et al., 2016) are based on manually developed rules, which heavily rely on expert knowledge, and have high maintenance costs. Moreover, as table size grows, these methods may encounter combinatorial explosion issues. Recently, machine learning-based methods have shown notable progress in enhancing recommendation accuracy and scalability. (Luo et al., 2018) employed expert rules for initial visualizations, and evaluated them using a trained classifier. (Li et al., 2021; Hu et al., 2019) formulated the visualization recommendation challenge as a series of classification tasks, learn to predict labels for various design choices, while (Zhou et al., 2020; Dibia and Demiralp, 2019) cast the challenge as a sequence generation problem. The former utilized a Deep Q-Network (DQN) for selecting action tokens to fill the predefined chart templates, while the latter employed a seq2seq model to autoregressively generate JSON-encoded charts. However, these approaches are limited to learning the existing knowledge within datasets, and lack sufficient exploration of multifaceted table features, such as table patterns and structures, leading to a limited comprehension of complicated tables. Recently, some researchers (Dibia, 2023; Ko et al., 2024;
Cheng et al., 2023; Maddigan and Susnjak, 2023) leveraged LLMs to investigate the visualization recommendation task. They divided the visualization generation process into several subtasks, exploring appropriate discrete prompts for each subtask to maximize the capabilities of the larger model. Different from prior arts, in our work, we enrich the extensive pre-trained knowledge of LLMs with multi-level prompt, to fully extract the semantics within table data from multiple dimensions.

2.2 Prompt Tuning

With continuous parameter scaling of pre-training models, fine-tuning entire models for downstream tasks becomes daunting due to inefficiency and potential catastrophic forgetting (Pfeiffer et al., 2021; Kan et al., 2023). Thus, Prompt Tuning (Lester et al., 2021), which prepends learnable continuous prompts to input text while keeping model parameters fixed, has achieved success across various domains. However, directly using this method to complex table datasets results in substantial information loss. In this paper, we use multi-level prompts to integrate table information, enhancing the efficiency of utilizing prior knowledge.

3 Preliminaries

3.1 Problem Formulation

We formulate the visualization recommendation task as a text generation problem. Consider a labeled training dataset \( D = \{(T, V)_i\}_{i=1}^{\vert D \vert} \) where \( T_i \) represents a serialized table and \( V_i = v_1^i, v_2^i, \ldots v_l^i \) is a concise json description of chart with length \( l_i \), which encapsulating key information for defining a chart. Our goal is to generate \( V_i \) in an auto-regressive way. Based on Prompt Tuning (Lester et al., 2021), we learn a prompts group \( P \) that have multi-level prompts, with a module \( G \) designed to dynamically produce prompts \( P \). We train our model by maximizing the likelihood of generating the target sequence, as follows:

\[
\max_{P, G} p_{\theta} \prod_{i=1}^{\vert D \vert} \prod_{j=1}^{l_i} \left( v_j^i \mid v_1^i, v_2^i, \ldots v_{j-1}^i, T_i, P, G \right).
\]

(1)

Here parameters \( \theta \) of LLM remain frozen, only the prompts group \( P \) and \( G \) are updated.

2.2.1 Table Linearization

As demonstrated in (Suadaa et al., 2021; Zhang et al., 2020), the representation of data in table form has a significant impact on generation performance. In this paper, we adopt a template-based linearization approach, transforming tables into flat string representations to enhance their compatibility with the internal structures of LLMs. Formally, given a table \( T \), it is represented as:

\(< \ header_{1,1} > is < value_{1,1} >, \cdots < header_{i,j} > is < value_{i,j} >, \cdots. \)

3.2 Data Serialization

3.2.2 Chart Mapping

We employ a JSON-based template to encode chart data similar to (Poco and Heer, 2017; Satyanarayan et al., 2017). The template encapsulates the key information necessary for defining a chart, including the visualization types and the arrangement of x-/y-axes. When using tools like Echarts or Plotly, our recommended attributes allow users to easily create chart images while giving them the freedom to adjust or add attributes to suit their specific analysis needs or presentation preferences if needed. The details of our template are as follows:

\[
\text{where each object in \text{“y”} and the objects of \text{“x”} form a visual trace.}
\]

4 Methodology

In this section, we will introduce the proposed framework in detail. As shown in Figure 2, our framework consists of four primary prompts, namely \textit{general}, \textit{instance}, \textit{column} and \textit{cluster} level prompts, to inject table insights into the LLM. Technical details of each component are discussed in the following subsections, step-by-step.

4.1 General-level Prompt Generation

To encompass the overall information of the table dataset and enhance generalization ability, we introduce a soft prompt \( P_{\text{gen}} \) as (Lester et al., 2021) do, which is uniformly applied across all instances. This general prompt ensures that the model comprehensively understands table dataset. It also
facilitates sharing and integrating information from other level prompts.

4.2 Instance-level Prompt Synthesization

Since tables target for visualizing data across diverse domains, there are inherent differences among tables within the dataset. Obviously, a single general prompt is insufficient to fully adapt to these complicated variations of tables. In response, we introduce an instance-aware prompt capable of dynamically capturing the table-specific features. In detail, we first synthesize a style vector that reflects the characteristics of different types of charts, named style query. Then, we use this style query to project information from the serialized table embedding into the chart space. In this way, the style query facilitates the extraction of fine-grained information from tables across various dimensions, enhancing the table’s perceptibility to charts. Moreover, projecting table information into chart space narrows down the search space for the LLM, thereby reducing the likelihood of producing hallucinations.

4.2.1 Style Query Generation

To ensure that the style query adequately represents the stylistic characteristics of each chart type, our style query is composed of two parts: hard codes and soft codes. Hard codes capture the inter-class features among different chart types, while soft codes are employed to explore the intra-class features within each chart type.

**Hard Codes Generation.** Since chart images have explicit labels (e.g., bar charts, pie charts), enabling distinction of different chart styles and further guiding chart generation. Thus, we employ a supervised contrastive learning approach that trains a projector \( P \) on labeled chart images, enabling it to learn the distinct representation of each chart style.

The detailed process is shown in Figure 2 (a). During training, we first feed a chart image \( i \) alongside its augmented version \( i^+ \) (e.g., rotation, grayscale), as well as other batch samples, into a frozen pre-trained image encoder to obtain visual embeddings. These extracted embeddings are then fed into a bottleneck architecture projector \( P \), which maps them to vectors that reflect chart style.

Then, we utilize supervised contrastive learning loss on generated vectors to train the projector \( P \). Drawing inspiration from MOCO (He et al., 2020), we maintain a continuously updated memory bank for each chart type. In our method, positive samples for a given instance include its augmented versions and same-type instances within batch and memory banks. In contrast, negative samples encompass instances of different chart types within the same batch and memory bank, ensuring that the model
learns to distinguish different chart categories.

As shown in Figure 2 (b), after the contrastive learning phase, we reprocess the visual embeddings of images through projector $P$ to obtain style features. To obtain an overall stylistic representation for each chart type, we categorize these features by chart type and perform an averaging operation within each category. In this way, we obtain a stylistic representation set $\{q_{m}^i\}_{i=1}^m$, named hard codes, where $m$ represents the number of chart types.

**Soft Codes Generation.** Even within the same chart category, there are many subtle variations. Relying solely on hard codes could lead to overlooking significant intra-class differences among chart types, limiting the model’s ability to detect subtle stylistic differences between charts. To more effectively explore the intra-class features among charts, we introduce $m$ trainable soft style codes $\{q_s^i\}_{i=1}^m$, which are prepended to the hard codes to provide a more comprehensive representation of the chart space.

Now the style query can be defined as follows:

$$Q_s = \{[q_s^i; \hat{q}_i]\quad i = 1, 2, ..., m.$$  

Here $[::]$ denotes the concatenation operation. Consequently, the style query $Q_s$ captures intra- and inter-class characteristics of each chart type simultaneously, which will be updated during training, allowing a better perception of input table.

### 4.2.2 Generating Instance-level Prompts with Style Query

We then use style query to guide instance-level prompt generation through a structure called Style Controller. As depicted in Figure 2 (e), we first employ a mapping network $M$ to align dimension and modality between $Q_s$ and the serialized input embedding $T_r$. For efficiency, $M$ consists of down and up projection layers, with a nonlinear layer situated between them. Then we adopt two attention layers: the self-attention over $Q_s$, followed by cross-attention from $T_r$, as follows:

$$Q_s = M(Q_s),$$

$$Q_s = SelfAttn(Q_s),$$

$$S = CrossAttn(Q_s, T_r).$$

The $S = \{s_1, s_2, ..., s_m\}$ represents features extracted by style query from $m$ spaces based on the distributional information of table embedding. Intuitively, self-attention layer enables the model to effectively process and integrate contextual information from style query. While the cross-attention layer can dynamically emphasize the essential chart-related information within a table through style query. Finally, we generate the instance-level prompt $P_{ins}$ by averaging the $s_i \in S$.

### 4.3 Cluster-level Prompt Generation

When two tables exhibit similar messages, they are likely to be represented by similar charts. Thus, using the shared information within table pattern is crucial for comprehending table representations and making chart recommendations. The diverse features of table data, such as trends, entropy, and variance, provide a comprehensive overview of table data, thus play a crucial role in identifying distinct table patterns. Thus, we extract various table features and cluster them into $k$ different table patterns. Each table schema has its own shared prompt, resulting in a cluster prompts pool $\{p_k^c\}_{k=1}^k$. Correspondingly, for a given table $T \in C_i$, its cluster prompt can be directly set as $p_k^c$.

### 4.4 Column-level Prompt Generation

The previous discussion emphasized the overall semantic understanding of the table. However, since the selection of chart type and the arrangement of x/y-axes are fundamentally based on column data, a detailed understanding of each column’s role and characteristics within the table is also critical for recommending appropriate charts. To this end, we introduce column-level prompt that designed to encapsulate column structure information within table, facilitating intra-column understanding and cross-column analysis.

As shown in Figure 2 (d), following (Chen et al., 2022), we first encode a given table $T$ on a per-cell basis to retain structure. Specifically, for each cell in $T$, we concatenate the cell’s header and value, and query the LLM to obtain embeddings. To enhance representation at the column level, we introduce a unique soft prompt for each column, which is preprended to the embeddings of cells within the same column, yielding a refined, column-centric table structure representation $E \in \mathbb{R}^{m \times n \times s \times d}$, with $s$, $m$ and $n$ denoting sequence length, number of rows and columns, respectively.

Next, we employ a two-layer attention structure, the first is a cell-wise layer that focus on individual cells, while the second operates on single column to explore column semantics:
\[ E_0 = E + E_{cpe}, \]
\[ \hat{E}_0 = \text{Linear}(\text{Self-Attn}(E_0)) + E_0, \]
\[ E_1 = \frac{1}{s} \sum_{i=1}^{s} \hat{E}_0[:,:,i,:], \]
\[ \hat{E}_1 = \text{Self-Attn}(E_1), \]
where \( E_{cpe} \in \mathbb{R}^{s \times d} \) is the cell text position embedding, which will be updated during training. In this way, the \( \hat{E}_1 \) can grasp the column structure information inside the table.

### 4.5 Prompt Integration for Reprogramming LLM

After obtaining the general, instance, and cluster level prompts, we use a shallow network \( V \) with a skip connection to reparameterize them. Subsequently, the reparameterized prompts are concatenated:

\[ P' = [V(P_{gen}); V(P_{ins}); V(P_{clu})], \]
\[ V(P) = \phi(P) + P, \]

where \( \phi \) is a simple linear layer. Furthermore, a self attention layer is applied to enable information sharing among the prompts:

\[ P_{pre} = \text{Self-Attn}(P'). \]

The refined prompts \( P_{pre} \in \mathbb{R}^{l_p \times d} \) are concatenated with the serialized table embedding \( T_r \in \mathbb{R}^{l_s \times d} \), where \( l_p \) denotes the prefix length, \( l_s \) denotes the input sequence length. They are then fed into the LLM to obtain the last hidden states \( H \). After that, we apply a cross-attention layer, using \( H \) as a query, to integrate the structured and unstructured information from the entire table:

\[ E_2 = \hat{E}_1 + E_{tpe}, \]
\[ H = \text{Cross-Attn}(H, E_2) + H, \]

where \( E_{tpe} \in \mathbb{R}^{m \times n \times d} \) is the table position embedding that provides row and column information for tables. This obtained hidden state will replace the original one to estimate probability of next word.

### 5 Experiments

#### 5.1 Settings

**Datasets** Due to inaccessibility of the Plotly Corpus, a publicly available dataset collected by (Hu et al., 2019), we independently re-extracted a substantial dataset of 443,526 public visualization pairs from the Plotly community feed via the Plotly REST API\(^1\). Following rigorous data cleaning, the refined dataset contains 116,528 visualization pairs across 439,001 columns, including 4,091(3.51%) pie, 36,576(31.38%) line, 29,740(25.52%) scatter, 6,269(5.38%) histogram, 29,641(25.43%) bar and 10,211(8.77%) box charts. More details are in Appendix A.

**Baselines** To evaluate the performance of our HTP framework, we compare our method with following baselines: (1) VizML (Hu et al., 2019), (2) KG4Vis (Li et al., 2021), (3) MultiVision (Wu et al., 2021), (4) Data2Vis (Dibia and Demiralp, 2019), (5) Table2Charts (Zhou et al., 2020), (6) DeepEye (Luo et al., 2018). More details are in Appendix B.

**Metrics** Referring to VizML, we employ Accuracy to evaluate whether models can recommend correct design choices, including: XY (whether the field encoded on the X-axis and Y-axis is correct), chart type (whether the correct chart type is recommended) and overall (both XY-axis and chart type are considered). Due to the variations among the above baseline models in calculating metrics, with some evaluations based on individual fields and others on entire tables, for fair comparisons, we report the metrics at the both level. More details are in Appendix C.

**Implementations** We randomly split the dataset into three parts: 80% for training, 10% for validation, and 10% for testing. For contrast learning, we use CLIP (Radford et al., 2021) visual encoder to extract features of input images. For our chart generation model, we primarily utilize the pre-trained Bloom (Workshop et al., 2022) model with 1.1B parameters as the backbone. We use K-Means (MacQueen et al., 1967) to cluster table features. Additional details are provided in Appendix D. \(^2\)

#### 5.2 Comparison with Baselines

The performance comparison of our HTP and baselines is presented in Table 1, in which the mean and standard deviation of all metrics are obtained through three random runs. We have the following observations: (1) Our HTP consistently achieves the best performance in most of the evaluation metrics, with 25.4% and 32.8% absolute improvements of overall accuracy in field level and table

\(^1\)https://api.plot.ly/v2/

\(^2\)We make the code and dataset of HTP available at: https://github.com/PaddlePaddle/PaddleSpatial/tree/main/research/HTP
To verify the effectiveness of each design in our model, we conduct a comprehensive comparison between HTP and the following methods at different scales.

**5.3 Ablation Analysis**

To verify the effectiveness of each design in our model, we further compare HTP with six variants:

- **w/o-CLUP** removes the cluster-level prompt.
- **w/o-TSI** removes the table structure information (include the column-level prompt).
- **w/o-COLP** removes the column-level prompt.
- **w/o-GP** removes the general prompt.
- **w/o-INSP** removes the instance-level prompt.
- **w/o-SSC** removes the soft codes, using only hard codes as source style query.

According to the results shown in Figure 3, HTP outperforms all its variants, proving the significance of our special designed prompts. Specifically, w/o-INSP and w/o-TSI underperform other variants, highlighting the need for dynamically generating customized prompts to capture fine-grained information and the internal structure of tables. Meanwhile, we can see w/o-INSP has a significant performance degradation on the chart type prediction, proving that instance-level prompt can effectively obtain perceptual information between table and different types of charts. Besides, we observe a decrease when removing the cluster-level prompt, which demonstrates the importance of sharing information within table patterns (w/o-CLUP). In addition, the performance degrades if the soft codes is not used, suggesting a comprehensive understanding of chart representation through capturing intra-class connections is crucial for enhancing model performance. Further, removing the table-column prompt results in a notable performance decline, which verifies the importance of understanding table structure and enhancing column-level insights.

**5.4 Comparison with Adaptation Methods across Model Scales**

To evaluate the effectiveness and adaptability of our model, we conduct a comprehensive comparison between HTP and the following methods at differ-
Table 2: Performance comparison between HTP and adaptation methods at different model scales. The best results except Fine-Tuning are in **bold**, and the second are underlined.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>Field Level</th>
<th>Table Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XY</td>
<td>Chart Type</td>
<td>Overall</td>
</tr>
<tr>
<td>Bloom-Small</td>
<td>Fine-Tuning</td>
<td>0.953</td>
<td>0.745</td>
</tr>
<tr>
<td>(560M)</td>
<td>Prompt Tuning</td>
<td>0.899</td>
<td>0.550</td>
</tr>
<tr>
<td></td>
<td>Prefix Tuning</td>
<td>0.924</td>
<td>0.592</td>
</tr>
<tr>
<td></td>
<td>LoRA</td>
<td>0.927</td>
<td>0.624</td>
</tr>
<tr>
<td></td>
<td>HTP</td>
<td><strong>0.902</strong></td>
<td><strong>0.714</strong></td>
</tr>
<tr>
<td>Bloom-Medium</td>
<td>Fine-Tuning</td>
<td>0.940</td>
<td>0.782</td>
</tr>
<tr>
<td>(1.1B)</td>
<td>Prompt Tuning</td>
<td>0.876</td>
<td>0.682</td>
</tr>
<tr>
<td></td>
<td>Prefix Tuning</td>
<td>0.912</td>
<td>0.746</td>
</tr>
<tr>
<td></td>
<td>LoRA</td>
<td>0.915</td>
<td>0.758</td>
</tr>
<tr>
<td></td>
<td>HTP</td>
<td><strong>0.928</strong></td>
<td><strong>0.780</strong></td>
</tr>
<tr>
<td>Bloom-Large</td>
<td>Fine-Tuning</td>
<td>0.950</td>
<td>0.832</td>
</tr>
<tr>
<td>(3B)</td>
<td>Prompt Tuning</td>
<td>0.817</td>
<td>0.670</td>
</tr>
<tr>
<td></td>
<td>Prefix Tuning</td>
<td>0.923</td>
<td>0.841</td>
</tr>
<tr>
<td></td>
<td>LoRA</td>
<td>0.936</td>
<td>0.813</td>
</tr>
<tr>
<td></td>
<td>HTP</td>
<td><strong>0.938</strong></td>
<td><strong>0.850</strong></td>
</tr>
</tbody>
</table>

Figure 4: Performance comparison between different prompt initialization methods.

Figure 5: Parameter sensitivity of the number of table pattern clusters $K$.

5.5 Parameters Analysis

5.5.1 Prompt Initialization

We explore the impact of soft prompt initialization in our study. Our investigation focuses on two distinct initialization strategies: (1) **Random-based Initialization**. Soft prompts are generated by randomly initializing their embeddings. (2) **Sample Vocabulary-based Initialization**. We sample high-frequency word chunks from the training set dictionary, concatenate them in order of frequency, then trim to match the prompts’ length. We use the embeddings of the trimmed version as initial values. As shown in Figure 4, we can see that HTP is robust to the prompt initialization method, achieving comparable results with both initialization choices.
5.5.2 Number of Table Patterns Clusters $K$

As depicted in Figure 5, more clusters of table pattern, namely increasing $K$, enables HTP to capture more complex and diverse table pattern information. However, if $K$ gets too large, there may be insufficient corresponding table groups in the table dataset to support cluster representation learning, and some superfluous clusters may introduce noise and undermine performance instead.

5.6 Comparison with GPT-4

Since the method we propose involves inputs (tables) and outputs (chart codes) that are both in text format, without any image inputs or outputs, to compare the performance of our HTP and GPT-4, we conduct an experiment with small data batches using GPT-4 (non-vision). We randomly selected 150 samples to conduct the experiment. As shown in Table 3, our HTP consistently outperforms GPT-4 across all tasks, with a particularly significant margin of excellence in chart-type recommendation tasks. This suggests that HTP demonstrates outstanding ability in understanding and recommending visualizations that are most suitable for the specific structure of the data at hand.

<table>
<thead>
<tr>
<th>Model</th>
<th>Field Level</th>
<th>Table Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XY Chart Type Overall</td>
<td>XY Chart Type Overall</td>
</tr>
<tr>
<td>GPT-4</td>
<td>0.832 0.403 0.385</td>
<td>0.727 0.453 0.427</td>
</tr>
<tr>
<td>HTP</td>
<td>0.881 0.722 0.683</td>
<td>0.841 0.753 0.693</td>
</tr>
</tbody>
</table>

Table 3: Performance comparison between HTP and GPT-4.

Acknowledgements

This work was supported in part by the grants from National Natural Science Foundation of China (No.62222213, U22B2059, 62072423, 92370204).

References


Daniel Perry, Bill Howe, Alicia Key, and C. Aragon. 2013. Vizdeck: Streamlining exploratory visual analytics of scientific data.


Appendix

A Data Cleaning

We developed our dataset using a data cleansing pipeline, outlined as follows:

(1) Format Verification. Since the Plotly community stores table and chart data in json format, we will initially filter out data that cannot be parsed in json format.

(2) Completeness Assessment. Data samples lacking either non-empty source table data or chart data are discarded. Additionally, if data columns corresponding to the x and y axes in the chart data are missing in the table data, if table data contains empty columns, or if chart data lacks specified chart type or axes, the sample is excluded.

(3) Data Deduplication. In the Plotly community, each visualization pair is uniquely identified by a “fid”. When encountering multiple data entries with the same “fid”, only the first one is retained. Since many tables are slight modifications of each other, we calculate their basic features, integrate these values into a single feature identifier, and retain only the first entry for each set of samples sharing this feature identifier.

B Baselines

To evaluate the proposed framework, we adopt six baselines for comparison. Here are the descriptions of these baselines:

- VizML (Hu et al., 2019), which formulates visualization recommendation task as a series of classification challenges, deploying distinct Neural Network-based models for each classification task. We train VizML in our dataset on its Mark Type task (chart type prediction task and Is on X-axis or Y-axis task). In our training process, we divide the dataset by tables, meaning all columns from the same table are assigned to the same set.
- KG4Vis (Li et al., 2021), which leverages knowledge graphs to recommend from dataset-visualization pairs.
- MultiVision (Wu et al., 2021), which design two scoring network for recommending single and multiple-view visualizations. Because we do not recommend multiple-view visualizations, only the first network is used to train on our dataset. Specially, due to the incomplete disclosure of all training parameters, we set the batch size to 4096 and the learning rate to 3e-3, conducting training over 30 epochs. Because our dataset contains six types of charts, so we set the initialization parameter num_class of the ChartTypeLSTM model to 6. The remaining model hyperparameters are consistent with those in the source code.
- Data2Vis (Dibia and Demiralp, 2019), which leverages an LSTM-based neural translation model to generate json encoded visualizations in an autoregressive way. Following Data2Vis, for each table-chart pair, three training samples are generated by sampling three rows from table (three different data rows with the same encoded chart), resulting in a total of 279, 609 pairs which are used for training. The vocabulary sizes of source and target are 95 and 38.
- Table2Charts (Zhou et al., 2020), which use deep Q-Network to fill the predefined chart templates by estimating the next token or action. Specifically because the chart templates defined in Table2Charts—scatter, line, bar, and pie match the types of charts in our dataset, we retain only these four types of data for training.
- DeepEye (Luo et al., 2018), which provides two public models (ML and rule-based) without training scripts. Thus, we evaluate its models on our test set and report the best results between two models.

For all baselines, all hyperparameter settings were based on the values reported in the original paper for optimal results, unless otherwise specified.

C Evaluation Details

Let $N$ denote the total number of tables, $C_i$ denote the number of columns in table $i$, correct$^{ij}_{xy}$ and correct$^{ij}_{type}$ as binary indicators for column $j$ in table $i$, where correct$^{ij}_{xy}$=1 if the xy axis allocation
for column $j$ is correct and 0 otherwise. Similarly, $correct_{xy}^{ij} = 1$ if the chart type for column $j$ is correctly predicted, and 0 otherwise.

At the table level, when calculating metrics, the entire table is treated as a single unit; a prediction is only considered correct if the X-Y axis allocation and chart type for all columns within the table are accurately predicted, as follows:

$$Acc_{xy} = \frac{\sum_{i=1}^{N}(\prod_{j=1}^{C_i} correct_{xy}^{ij})}{N},$$

$$Acc_{chart\ type} = \frac{\sum_{i=1}^{N}(\prod_{i}^{C_i} correct_{ij}^{type})}{M},$$

$$Acc_{overall} = \frac{\sum_{i=1}^{N}(\prod_{j=1}^{C_i} (correct_{xy}^{ij} \& correct_{ij}^{type}))}{M}. \quad (7)$$

While at the field level, we calculate metrics on a per-column basis:

$$M = \sum_{i=1}^{N} C_i, Acc_{xy} = \frac{\sum_{i=1}^{N}(\prod_{j=1}^{C_i} correct_{xy}^{ij})}{M},$$

$$Acc_{chart\ type} = \frac{\sum_{i=1}^{N}(\prod_{i}^{C_i} correct_{ij}^{type})}{M},$$

$$Acc_{overall} = \frac{\sum_{i=1}^{N}(\prod_{j=1}^{C_i} (correct_{xy}^{ij} \& correct_{ij}^{type}))}{M}. \quad (8)$$

## D Implementations

To prevent bias towards imbalanced data, we randomly split the dataset by chart type, allocating 80% for training, 10% for validation, and 10% for testing for each chart category. We run all experiments on NVIDIA RTX A100 GPUs, and use the Adam (Kingma and Ba, 2015) optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ to optimize models. For contrast learning, we adopt CLIP’s (Radford et al., 2021) visual encoder to extract the global features of input images. The temperature parameter, memory bank size, batch size and learning rate are set to 0.07, 2048, 1024, and 9e-4, respectively. We employ K-Means (MacQueen et al., 1967) to cluster table features, to minimize the impact of random initialization of initial cluster centers as much as possible, we execute the algorithm five times and select the iteration with the lowest Sum of Squared Errors (SSE) as the final result. For our chart generation model, we primarily utilize the pre-trained Bloom (Workshop et al., 2022) model with 1.1 billion parameters as our backbone. For training our chart generation model, we set the learning rate, batch size to 1e-5 and 4. The length of general-level, instance-level, column-level, cluster-level prompts are set to 20, 100, 1, and 15. In addition to parametric experiments, we fixed the number of clusters for cluster prompt at 8 and the prompt initialization method to Sample Vocabulary-based Initialization. In the generation stage, we adopt top-$p$ sampling as the default decoding method with a temperature of 0.1 and a top-$p = 0.75$.

## E Table Features for Clustering

We use table features that introduced in VizML (Hu et al., 2019) for clustering. Following VizML, we first extract 81 single-column features, comprising both 50 continuous features and 31 categorical features. These features are categorized into four main groups: the data type of the column (Types), statistical characteristics of the column’s values such as distribution and outliers (Values), the column’s name (Names), and the column’s row count (Dimensions). To describe the relationship between pairs of columns, we employ 30 pairwise-column features. In the final step, we utilize 16 aggregation functions to combine both pairwise-column and single-column features, yielding 841 dataset-level features for clustering.

## F Discussion about Permutations

To investigate the impact of the permutations of table rows and columns on the results, we conducted relevant experiments. The results indicate that the permutations of tables can have a certain negative impact on the inference results of our model. We will explore ways to mitigate this issue in our future work.

## G More Results about Ablation Analysis

Due to space constraints and the fact that trends at both the table level and field level are largely similar, we have chosen to present only the table level results in the main text. For completeness, we also include the field level results in Figure A1. The analysis of ablation studies at the field level follows a similar pattern to that at the table level.
H Image Dataset

We collect 125,121 chart images from the public web. These include 28100 bar images, 9838 box images, 7319 histogram images, 38725 line images, 37215 scatter images and 3924 pie images.