Dataset Discovery via Line Charts

Daomin Ji, Hui Luo, Zhifeng Bao, Shane Culpepper
daomin.ji@student.rmit.edu.au,huil@uow.edu.au,zhifeng.bao@rmit.edu.au,s.culpepper@uq.edu.au

ABSTRACT

Line charts are a valuable tool for data analysis and exploration, distilling essential insights from a dataset. However, access to the underlying dataset behind a line chart is rarely readily available. In this paper, we explore a novel dataset discovery problem, dataset discovery via line charts, focusing on the use of line charts as queries to discover datasets within a large data repository that are capable of generating similar line charts. To solve this problem, we propose a novel approach called Fine-grained Cross-modal Relevance Learning Model (FCM), which aims to estimate the relevance between a line chart and a candidate dataset. To achieve this goal, FCM first employs a visual element extractor to extract informative visual elements, i.e., lines and y-ticks, from a line chart. Then, two novel segment-level encoders are adopted to learn representations for a line chart and a dataset, preserving fine-grained information, followed by a cross-modal matcher to match the learned representations in a fine-grained way. Furthermore, we extend FCM to support line chart queries generated based on data aggregation. Last, we propose a benchmark tailored for this problem since no such dataset exists. Extensive evauation on the new benchmark verifies the effectiveness of our proposed method. Specifically, our proposed approach surpasses the best baseline by 42.3% and 54.2% in terms of *prec@50* and *ndcg@50*, respectively.

PVLDB Reference Format:
Daomin Ji, Hui Luo, Zhifeng Bao, Shane Culpepper. Dataset Discovery via Line Charts. PVLDB, 14(1): XXX-XXX, 2020.doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://anonymous.4open.science/r/DDLC-0FCA/.

1 INTRODUCTION

Dataset discovery [2, 5, 11, 24] involves the identification, assessment and selection of datasets from a large dataset respository based on the specific requirements and objectives of a user. It has a pivotal role in data-driven workflows, and has a profound influence on the quality and success of subsequent data analyses. Depending on the user input, existing studies can be broadly divided into two categories: 1) keyword-based dataset discovery [5], which aims to find datasets whose contents are the most relevant to a set of given keywords; 2) unionable or joinable dataset discovery [2, 11, 24], which aims to find datasets (tables) that can be joined/merged with a user-provided table.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

While keywords and tables serve as valuable inputs for dataset discovery, users may also seek alternative informative intermediaries. To the best of our knowledge, this work, introduces a novel dimension to dataset discovery by exploring the use of line charts as a means to identify relevant datasets. Line charts, which are common visual representations of data, have a pivotal role in effective communication and interpretation of information in data analysis tasks. The objective is to unearth datasets capable of generating a line chart akin to an exemplar provided by the user.

The problem of dataset discovery via line charts offers broad applications across various domains, enhancing data exploration and decision-making processes in scenarios where users aim to uncover datasets aligning with the raw information behind a line chart. For example, in rumor analysis, identifying original data sources from widely disseminated line charts aids in verifying authenticity [6]. In healthcare, it benefits doctors seeking relevant data for treatment decisions or participant recruitment in drug trials, given the extensive use of line charts, such as electrocardiograms, in medical content [21, 23]. In stock analysis, analysts can explore historical data corresponding to stock trend line charts for forecasting [26].

Our work draws inspirations from and contributes to two related research fields: databases and visualization. In the field of databases, it aligns with the well-established practice of dataset discovery [2, 11, 24], as discussed earlier. In the field of visualization, it is similar to chart search [30, 34, 39], which identifies visually similar charts using an exemplar, and benefits other foundational tasks such as visualization recommendation [33]. The above research examples primarily focus on identifying relationships or similarities using a single data modality (e.g., table-to-table and image-to-image) or between two closely related modalities (e.g., text-to-table). Our research is unique as it is centered on two distinct data modalities: structural tables and image data, providing a novel contribution to the exploration of relationships and similarities across disparate data modalities.

More specifically, the novelty of dataset discovery via line charts can be demonstrated as follows. First, we avoid imposing rigid constraints on the exemplar line chart input, requiring only two essential visual elements–namely, lines and the y-ticks. Additionally, the line chart can feature either a single line or multiple lines. More importantly, we extend our scope to encompass line charts depicting visual outcomes derived from *data aggregation operations* applied to a dataset. This extension is significant as data aggregation commonly serves as an operator to summarize data.

When comparing our problem against existing dataset discovery problems, dataset discovery via line charts faces two unique challenges:

(1) **Challenge 1: Cross-modal Relevance.** Defining an appropriate relevance metric between an exemplar line chart and a candidate dataset significantly impacts the quality of discovered datasets. In this bimodal task, utilizing both image and tabular data, it deviates from the majority of existing dataset

discovery methods [2, 24, 44], which predominantly focus on a single modality.

(2) **Data Shift between Aggregated Data and Original Data.** A line chart can be generated by aggregating data from the original dataset, introducing a shift gap between the aggregated and original data. This shift is further complicated by the multitude of valid data aggregation operators and the arbitrary aggregation window size. Therefore, it is critical to answer the following three questions to effectively handle aggregation-based queries in our problem: 1) How can we model data transformations created from a variety of different aggregation operators? 2) How can we capture the effects created by arbitrary aggregation window sizes? 3) How can we infer the most likely aggregation operator used for a particular line chart?

Our Contribution. In this paper, we have made the following contributions:

A Novel Cross-Modal Relevance Learning Model. To address the challenge of cross-modal relevance, we propose a novel model called Fine-grained Cross-modal Relevance Learning Model (FCM). FCM first employs a visual element extractor to extract essential visual elements, such as lines and y-tick values, from a line chart, which serve as key indicators of desirable datasets. Then, to match a dataset and a line chart, we propose two novel segment-level encoders to learn the representations of the line chart and the candidate dataset, preserving locality-based characteristics. Finally, a cross-modal macther is created to match them in a fine-grained manner.

Handling Aggregation-based Queries. To address the challenge of data shift between the original data and the aggregated data, we enhance FCM with three innovative components in the dataset encoder: (1) The transformation layer aims to learn non-linear transformations produced by data aggregation operations. (2) The hierarchical multi-scale representation learning layer, which adopts a tree structure to learn the dataset representation in a bottom-to-up way. Nodes at various levels of the tree encapsulate representations corresponding to different aggregation window sizes. Consequently, the top-level node offers a comprehensive representation, integrating information across these varying window sizes. (3) The Mixture-of-Experts [55] layer learns the distribution about different aggregation operators in order to automatically infer the most likely aggregation operation used in a line chart.

Efficiency. We improve the efficiency from two aspects: offline model training and online query processing. Specifically, for offline model training, we adopt a semi-hard negative selection strategy to accelerate convergence; For online query processing, we propose a hybrid indexing strategy based on interval tree [27] and locality sensitive hashing [35] to identify a small set of candidate datasets, which greatly reduces the number of relevance estimations required. (Sec. 6)

Benchmarks and Evaluation. As the first to explore dataset discovery via line charts, we establish a benchmark for evaluation based on Plotly [19]. This benchmark is publicly released in [1]. Extensive experiments demonstrate the effectiveness of our solution, when compared against the best baseline, the relative improvement of FCM is 42.3% and 54.2% in terms of *prec@50* and *ndcg@50*, respectively. (Sec. 7)

2 PROBLEM FORMULATION

We first describe key concepts used in this work, and then introduce the problem of dataset discovery via line charts and how we transform it into a problem of cross-modal relevance learning.

Dataset. In this work, we consider a dataset to be a table of N_C columns, namely $T = \{C_1, C_2, ..., C_{N_C}\}$. Each column C_i is expressed as a data series $C_i = (a_1, a_2, ..., a_{N_R})$, where N_R is the number of rows in T. Henceforth, we use "table" and "dataset" interchangeably.

Line Chart. A line chart V, employed as a visual representation, is a 2-D image presenting one or multiple data series across a continuous interval or time period. V consists of several visual elements, out of which two are essential in every line chart:

- *Lines* are the key visual elements in a line chart that demonstrate how the underlying data will change within a given period. We introduce the concept of "underlying data" later in this section. A line chart may contain M lines, denoted as $L = \{l_1, l_2, ..., l_M\}$.
- *Ticks* are the markings or values displayed along each axis, indicating the range corresponding to the points of each line.

Notably, the essential visual elements alone suffice for effective functioning of our approach. Several optional visual elements, like label and legend, are described in the technical report [1].

Underlying Data. The underlying data D is what a user aims to present in a line chart V, which contains M data series $D = \{d_1, d_2, ..., d_M\}$, each being a list of data points $d = (p_1, p_2, ..., p_{N_d})$. N_d denotes the number of data points in the data series. Each data series corresponds to one line, and each point p_i corresponds to a key-value pair (x_i, y_i), where x_i represents a timestamp and y_i represents its associated value. Furthermore, x_i could be a single time step such as "2023-11-20", or just an index such as "1", and y_i is a numerical value. Moreover, all the data series $d \in D$ will share the same values for x_i $(1 \leq i \leq N_d)$.

In practice, users usually generate the underlying data D from a dataset T that they wish to visualize. Usually, x-axis values and y-axis values of the underlying data D correspond to a column pair (C_i, C_j) in a dataset T. In some cases, x-axis values could be unspecified by the user and are automatically generated as an index (i.e., 1, 2, 3, ...) for the line chart. For simplicity, we also denote this case as (C_i, C_j), where $C_i = 1, 2, 3, ...$ Depending on whether data aggregation is applied, there are two different ways to generate the underlying data D from a column pair (C_i, C_j):

- Directly employ a column pair (C_i, C_j) to produce a data series d. Here, C_i and C_j correspond to the values on the x-axis and y-axis in the data series, respectively.
- Generate a data series d by applying a data aggregation operator into the column pair (C_i, C_j). This is widely used to summarize data over a given period. Specifically, we denote the set of all valid aggregation operators for (C_i, C_j) as $\mathcal{A} = \{\mathcal{A}_x, \mathcal{A}_y\}$, where $\mathcal{A}_x$ and $\mathcal{A}_y$ are valid operators for C_i and C_j, respectively. In this work, we focus on the following aggregation operators tailored for line charts: $\mathcal{A}_x = \{bin\}$ and $\mathcal{A}_y = \{avg, sum, max, min\}$.

Our focus is to find datasets (within a dataset repository) that are capable of generating a line chart similar to a line chart query. However, given the numerous ways to create a line chart from a dataset, it can be both time-consuming and potentially impossible to generate all possible line charts and then calculate the similarity between each of them and the line chart query.

To address this problem, one alternative is to assess the relevance between the underlying data D of a line chart query V and a candidate dataset T. By ensuring that there is a meaningful similarity between T and D, it is plausible that T can produce a line chart similar to V. We will discuss how to define a proper relevance score $Rel(D, T)$ between D and T later in Sec. 3.1. Assuming the existence of $Rel(D, T)$, dataset discovery using line charts is analogous to the top-k search problem described below:

Definition 1. ***Dataset Discovery via Line Charts****. Given a line chart query V that is generated from underlying data D and a repository of datasets $\mathcal{T} = \{T_1, T_2, ..., T_{|\mathcal{T}|}\}$, find a list of top-k datasets from $\mathcal{T}$ using a pre-defined relevance score $Rel(D, T)$.*

The top-k search problem has been extensively studied. However, a significant obstacle is the unavailability of D in the query stage, making it impossible to compute $Rel(D, T)$ directly. To overcome this issue, given the richness of features in V that represent D, our approach aims to learn an alternative relevance function, denoted as $Rel'(V, T)$, to approximate $Rel(D, T)$. Thus, our problem can be transformed into a *cross-modal relevance learning* problem.

Definition 2. ***Cross-modal Relevance Learning.*** *Given a training set of triplets $\mathcal{S} = \{(V_i, D_i, T_i)\}_{i=1}^{i=|\mathcal{S}|}$, where the line chart V_i is used for visualizing the underlying data D_i that is generated from the dataset T_i, the goal of cross-modal relevance learning is to learn a relevance function $Rel'_\Theta(V, T)$, such that*

$$argmin_\Theta \sum_{i=1}^{|\mathcal{S}|} |Rel'_\Theta(V_i, T_i) - Rel(D_i, T_i)|, \quad (1)$$

where Θ denotes parameters for the relevance function.

For simplicity, we will use $Rel'(V, T)$ and $Rel'_\Theta(V, T)$ interchangeably in the rest of this paper. We will introduce how we create such a training set $\mathcal{S}$ in Sec. 7.1.

3 KEY IDEA AND SOLUTION OVERVIEW

Our solution is to develop a machine learning model to learn $Rel'(V, T)$ as a proxy of $Rel(D, T)$. Given the absence of a clear definition for the relevance between a dataset and the underlying data of a line chart, we first present how to define $Rel(D, T)$. $Rel(D, T)$ will be used to identify informative training examples for model training. Next, we will describe the key idea on how to establish an ideal mapping between a line chart and a dataset, which guides the architectural design of our model to compute $Rel'(V, T)$. Last, we will provide a holistic overview of our proposed method.

3.1 A Proper Definition of $Rel(D, T)$

Since the underlying data D and the dataset T contain multiple data series, we define $Rel(D, T)$ in a bottom-up manner.

Low-level Relevance seeks to quantify the relevance $rel(d, C)$ between a column $C \in T$ and a data series $d \in D$. Each data series d consists of two sequences of values, x-values and y-values, while C is a single data series. When measuring the relevance between C and d, we ignore the x-values in the data series for two reasons: (1) Users may wish to identify historical data trends similar to the current data, so the x-values might be misleading in this case; (2) As described in Sec. 2, sometimes x-values are not provided in datasets and are generated automatically by the visualization tool. Thus, measuring $rel(d, C)$ can be reduced to measuring the relevance between the column C and the y-values of d. To achieve this, we use dynamic time warping (DTW) distance [8, 10, 14], which is widely used to deal with sequences of data that have varying lengths or temporal resolutions when measuring the distance between two data series. Note that there is previous work using Euclidean Distance (ED) to calculate the distance. However, since ED cannot capture data shifts, so we do not use it in this paper. Specifically, for a pair (d, C), the relevance score is defined as $rel(d, C) = \frac{1}{1+dist(d,C)}$, where $dist$ is the DTW distance function. Note that a larger DTW between d and C implies a lower relevance between the two datasets.

High-level Relevance seeks to quantify the relevance between D and T. To achieve this, it is crucial to establish the relationship between each column in T and each data series in D. Drawing inspiration from prior work on schema matching [11], we formulate our problem as a weighted maximum bipartite matching problem [49]. Here, each data series is regarded as a node in a bipartite graph $\mathcal{B} = \{\mathcal{V}, \mathcal{E}\}$ and $rel(d_i, C_j)$ is the weight of the corresponding edge (i, j) in $\mathcal{B}$. The objective is to identify a subset of edges, denoted as $\mathcal{B}' \subseteq \mathcal{B}$, which maximizes the weighted sum, subject to the constraint that no two edges in $\mathcal{B}'$ share a common node. Once $\mathcal{B}'$ is determined, we can establish the mapping between the data series in D and the columns in T. Then, the high-level relevance is computed as the sum of the weights of the selected edges, formulated as $\mathcal{B}'$: $Rel(D, T) = \sum_{(i,j)\in\mathcal{E}} rel(d_i, C_j)x_{ij}$, subject to 1) $\sum_{j:(i,j)\in\mathcal{E}} x_{ij} \leq 1, \forall d_i \in D$, and 2) $\sum_{i:(i,j)\in\mathcal{E}} x_{ij} \leq 1, \forall C_j \in T$, where $x_{ij} \in \{0, 1\}, \forall (i, j) \in \mathcal{E}$.

The above weighted maximum bipartite problem can be solved by Hungarian algorithm [28], which has a time complexity of $O(|\mathcal{V}|^3)$.

3.2 Key Idea

We first illustrate how to match a line chart with a candidate dataset in an ideal way, using the following example.

Example 1. *As illustrated in Fig. 2, the line chart V contains two lines and the dataset T has three columns. To match the chart with a dataset, we may want to know if each line in the chart can be generated from the column in the dataset. To resolve this, we first try to match each line and each column, finding that Line 1 matches column A exactly while both Column B and Column C do not match Line 2. However, when we divide Line 2, Column B, and Column C into four segments (shown in different colors and separated by dots), the initial three-quarters of Line 2 exactly align with Column B, but none of the segments in Column C correspond to any part of Line 2.*

This example underscores the importance of *locality matching* as well as fine-grained matching when attempting to match a line chart with a dataset. Specifically, the matching is performed at two different levels: (1) At the line-to-column level, the comparison between each line and each column is performed; (2) At the segment-level, the comparison between a line segment and a data segment is carried out. This fine-grained matching process guides the design of our model when computing $Rel'(V, T)$ – the model should have the ability to learn representations from a line chart and a dataset that can *preserve locality (segment-level) information and align the segments from a line chart with a column during the matching process.*

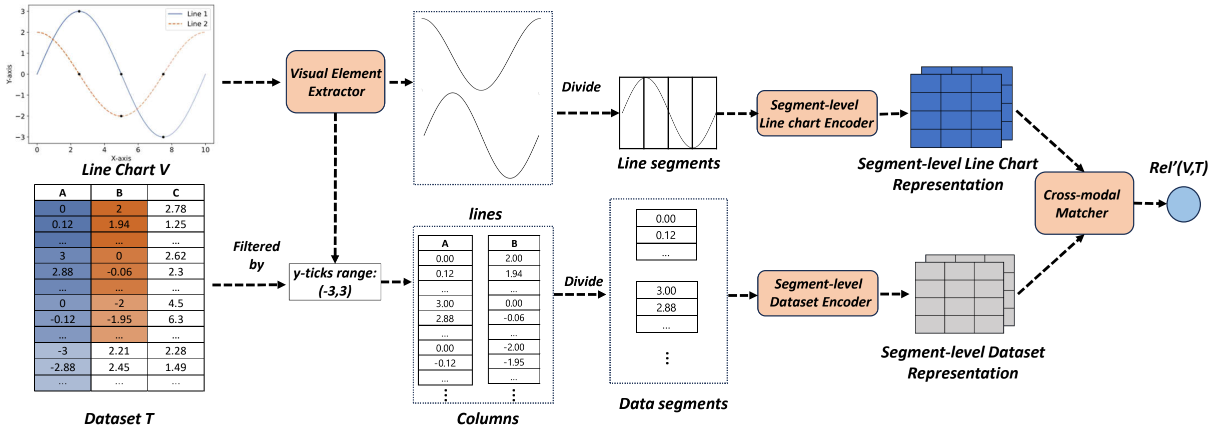

Figure 1: An overview of Fine-grained Cross-modal Relevance Learning Model (FCM).

3.3 Solution Overview

Based on this idea, we propose a novel Fine-grained Cross-modal Relevance Learning Model (FCM). Fig. 1 provides an overview of our model, consisting of four components:

- *Visual Element Extractor* extracts informative visual elements relevant to dataset discovery via line charts task. The lines and the y-axis ticks, extracted from a line chart V, serve as input to subsequent components: the lines are sent to an encoder to learn the representation of the line chart V; the y-axis ticks are used to filter columns from a candidate dataset T (Sec. 4.1).
- *Segment-level Line Chart Encoder* specializes in learning the representation of a line chart V at the segment level. Here, a segment pertains to a line segment, containing the local context information derived from a line (Sec. 4.2).
- *Segment-level Dataset Encoder* learns the representation of a candidate dataset T at the segment level. Here, a segment refers to a consecutive range of data points in a column (Sec. 4.3).
- *Cross-modal Matcher* learns a fine-grained alignment between a line chart and a dataset based on the learned representations at two levels, the line-to-column level and the segment level, and outputs a score to estimate the relevance (Sec. 4.4).

Extension to Handle Aggregation-based Queries. In practice, data aggregation (DA) operations are usually used when generating a line chart. To handle aggregation-based queries, we incorporate three innovative DA-related layers into the dataset encoder to learn a comprehensive and accurate representation capable of capturing data aggregation operations, as illustrated in Fig. 3 (Sec. 5).

Hybrid Indexing Strategy. While FCM is effective at estimating the relevance score between a line chart and a candidate dataset, directly applying it using a simple linear scan algorithm at the query stage is computationally expensive. Thus, a hybrid indexing strategy using an interval tree [27] and locality-sensitive hashing [35] is proposed to improve the search efficiency at the query stage. (Sec. 6)

4 FINE-GRAINED CROSS-MODAL RELEVANCE MODELING

4.1 Visual Element Extractor

As shown in Fig. 1, the first step of FCM is to extract informative visual elements from the line chart V. This serves as a pivotal step in FCM, conferring several advantages: 1) it allows the model to focus on pertinent visual elements, filtering out the influence of non-relevant ones; 2) by extracting each distinct visual element, such as individual lines from the line charts, a fine-grained matching process between the line chart and the prospective dataset becomes feasible, as detailed in Sec. 3.2. Specifically, for the etracted ticks, we utilize only the *y-ticks* as the essential visual elements, as they provide key information about the range of possible values. We do not leverage *x-ticks* in this task for two reasons: (1) our objective often involves the discovery of historical data analogous to the query, without necessitating specific constraints on the corresponding x-axis values within candidate datasets; (2) as previously outlined in Sec. 2, instances exist where the values on the x-axis are automatically generated and are a part of the original datasets.

To extract the two essential visual elements from V, we can resort to existing image segmentation methods [25, 40], which have been widely employed in computer vision. When applying image segmentation in our task, there are two options: 1) Direct utilization of a pre-trained large image segmentation model; 2) Training a segmentation model from scratch. However, when we tried to use SAM [25], one of the most powerful pre-trained image segmentation methods, to extract visual elements in a line chart, the results were not promising, achieving a mean average precision of only 0.45. One possible reason is that the training images used during the pre-training stage are much different from our case, i.e., line charts. Hence, we train our image segmentation method from scratch. However, we currently lack a dataset specifically tailored for segmenting objects within line charts. Creating such a dataset manually would be costly in terms of both labor and curation, as it would require annotating class information for every pixel in each image. To address this issue, we introduce *LineChartSeg*, the first dataset designed for line chart segmentation. *LineChartSeg* is generated by automatically labeling pixel-level information using visualization libraries, which offer insights into pixel rendering when generating a chart image. Furthermore, we propose a novel data augmentation method tailored to effectively train a line chart segmentation model (LCSeg).

LineChartSeg. We use Plotly [19], a vast real-world dataset extensively utilized in data visualization, to construct the *LineChartSeg* dataset. Plotly contains millions of tabular datasets, each associated with a visualization specification describing how to utilize the table columns for creating the visualization. Our approach generates a training example in *LineChartSeg* based on each (table, visualization specification) pair. In the image segmentation, a typical training example includes an input image requiring segmentation and a set of masks representing pixel-level class information. To create our training examples, for each table, we use the corresponding visualization specification to generate a line chart using *matplotlib* [22], a popular visualization library. In *matplotlib*, each visual element is described using an *artist* class, such as axis, label, and line. Using the *transdata* method in *matplotlib*, we can obtain the pixel coordinate location for each *artist*. This allows us to generate masks for the input image by assigning distinct colors to each visual element.

Data Augmentation for Segmentation Model Training. We utilize *LineChartSeg* to train our line chart segmentation model (LCSeg) using a Mask RCNN [16], a robust image segmentation model widely applied in computer vision. When training an image segmentation model, the incorporation of data augmentation methods is essential to achieve the full potential from each training example. However, conventional data augmentation techniques may not be suitable for our task as they can alter the semantic meaning of the chart. For instance, common methods such as *flipping* [16], which horizontally or vertically mirrors images, are not appropriate since flipping the chart image may distort key information, such as labels and ticks, diminishing informativeness.

To resolve this challenge, we propose a novel data augmentation method to train LCSeg. Our key idea is to perform data transformations on the original tabular datasets from which the line chart is originated. This new strategy not only preserves the integrity of the line chart but also enables the generation of line chart images that adhere closely to real data without modification, avoiding deviations from exemplars. Specifically, we propose the following data augmentations methods:

- *Reverse.* For each column $C = (a_1, ..., a_n)$ in a dataset, we reverse all data in the column to form a new column $C' = (a_n, ..., a_1)$.
- *Partitioning.* Each column $C = (a_1, ..., a_n)$ in a dataset is randomly partitioned at position n', to yield two columns $C'_1 = (a_1, ..., a_{n'})$ and $C'_2 = (a_{n'+1}, ..., a_n)$.
- *Down-Sampling.* Each column $C = (a_1, ..., a_n)$ in a dataset is down-sampled to contain c at a ratio of $\frac{1}{\rho}$, such that in the resulting column C', only one data point is retained for every ρ consecutive data points in C.

4.2 Segment-level Line Chart Encoder

The line chart encoder is designed to learn the representation of a line chart while preserving its segment-level information. Given a line chart V, we have used the visual element extractor (as described in Sec. 4.1) to extract each line from the chart. Each line l is then represented as a 2-D image, denoted as $I \in \mathbb{R}^{H\times W\times Q}$, where H and W represent the height and width of the image, and Q denotes the number of color channels (e.g., red, green, and blue). While color channels are vital in traditional computer vision tasks such as image classification [17] and object detection [16], they have minimal impact in our task, which focuses primarily on the shape of a line. To accomplish this, for each line l, we uniformly transform the chart into a greyscale image, denoted as $I_{grey} \in \mathbb{R}^{H\times W}$. This transformation not only filters out all color information, it also reduces the input data size by a factor of Q, hence improving processing efficiency.

As shown in the lower left section of Fig. 2, each line is divided into a sequence of small 2-D images of line segments $I_{grey} = \{z_1, z_2, ..., z_{N_1}\}$, $z_i \in \mathbb{R}^{H\times P_1}$, where P_1 denotes the width of each small image, and $N_1 = \frac{W}{P_1}$ is the resulting number of small images. To learn representations for these line segment images, we apply a Vision Transformer (ViT) [9, 15], which is a transformer-based architecture that encodes the image features. Each line segment image z_i is flattened into a 1-D vector of length WP_1, which is then transformed into a vector of length K using a trainable linear projection layer. Here, K denotes the embedding size.

Using the above procedure, a line image I is transformed into a sequence of embedding vectors. Then, the encoder applies a standard transformer to further encode the sequence, enabling an encoder to learn correlations between the line segments. The transformer encoder consists of J multi-head self-attention (MSA) and multi-layer perceptron (MLP) blocks, and processes a line l as follows:

$$\begin{aligned} u_0 &= [z_1; z_2; ...; z_{N_1}] + E_{pos}, && E_{pos} \in \mathbb{R}^{N_1\times K}, \\ u'_i &= MSA(LN(u_{i-1})) + u_{i-1}, && i = 1, ..., J, \\ u_i &= MLP(LN(u'_i)) + u'_i, && i = 1, ..., J \end{aligned} \tag{2}$$

Here, LN denotes the layernorm, a technique employed to normalize the input and expedite transformer convergence; E_{pos} are positional embeddings that are trainable parameters and encode positional information for the line segments in line l.

As a result, $u_J \in \mathbb{R}^{N_1\times K}$ is the representation for a line, where each row in u_J corresponds to the representation of a line segment. Note that there are M lines in a line chart. Therefore, for the entire line chart V, we combine all of the representations of the lines contained in V, leading to a final representation $E_V \in \mathbb{R}^{M\times N_1\times D}$ for the line chart V. For simplicity, we use $E_V[i]$ to denote the representation of the i-th line in V, and $E_V[i, j]$ to denote the representation of the j-th segment of the i-th line in V.

4.3 Segment-level Dataset Encoder

For each dataset T, before discussing the representation learning process, a crucial initial step involves leveraging information from the y-axis ticks in an image to determine the range of values of the data and filter columns in T. Subsequently, the dataset encoder focuses on learning a segment-level representation for each remaining column C. Specifically, each column C can be represented by a data series $C = (a_1, a_2, ..., a_{N_R})$ where a_i denotes the content of i-th cell in C. We partition C into N_2 segments, $C = (w_1, w_2, ..., w_{N_2})$, where each w_i contains $P_2 = \frac{N_R}{N_2}$ data points (cell values). Then, a transformer-based architecture can be applied to learn a representation for each data segment. In addition to preserving local semantics and facilitating fine-grained matching, this process also improves efficiency. Specifically, partitioning a column (of length N_R) into N_2

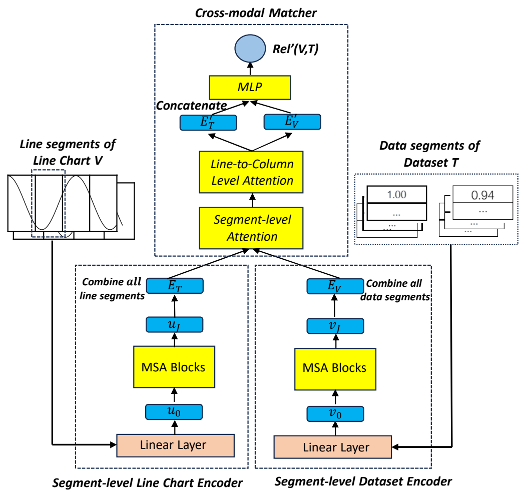
Cross-modal Matcher
Rel'(V,T)
MLP
Concatenate
E'_T
E'_V
Line-to-Column Level Attention
Segment-level Attention
Line segments of Line Chart V
Data segments of Dataset T
1.00
0.94
Combine all line segments
E_T
u_J
MSA Blocks
u_0
Linear Layer
Combine all data segments
E_V
v_J
MSA Blocks
v_0
Linear Layer
Segment-level Line Chart Encoder
Segment-level Dataset Encoder

Figure 2: The architecture of the segment-level encoders and the cross-modal matcher.

segments results in measurable memory and time savings – approximately a factor of $(P_2)^2$ – when computing self-attention, which is a crucial component in the transformer-based encoder since the time and space complexity of self-attention are quadratic w.r.t. the length of the input sequence.

The lower right of Fig. 2 shows the architecture of the segment-level dataset encoder. First, each data segment w_i is mapped to a vector v_0 of length K using a trainable linear projection layer. The output is then fed into a standard transformer to derive the final representation for each data segment of column C. This process is similar to the line chart encoding, and for brevity, we do not repeat those details here. Ultimately, the representation for each data segment across every column in T is combined into a final representation, denoted as $E_T \in \mathbb{R}^{N_C \times N_2 \times D}$, where N_C denotes the number of columns in T. Similarly, $E_T[m]$ denotes the representation of the m-th column in T, and $E_T[m,n]$ denotes the representation of the n-th segment of the m-th column of T.

4.4 Cross-modal Matcher

Given representations E_T and E_V, the cross-modal matcher estimates the relevance score $Rel'(V,T)$ at two levels: the line-to-column level and the segment level, as outlined in Sec. 3.2. To achieve this, we propose a hierachical cross-modal attention network (HCMAN) to implement the matcher. As shown at the top of Fig. 2, HCMAN has two levels of self-attention networks, each designed to capture the relevance between elements at their respective level.

Specifically, given two representations for a line chart and the target dataset, E_V and E_T, HCMAN matches each line segment representation $E_V[i,j]$ and each data segment representation $E_T[m,n]$ with a segment-level self-attention network (SL-SAN). SL-SAN first transforms each column segment representation or line segment representation into three distinct vectors: a query vector p_q, a key vector p_k, and a value vector p_v. Then, the relevance score between $E_V[i,j]$ and $E_T[m,n]$ is computed using $sim(p_q(E_V[i,j]),$ $p_k(E_T[m,n]))$, where sim is a scaled dot-product similarity function [50]. Depending on the relevance score returned, the line (column) representation $E'_V[i]$ ($E'_T[m]$) is reconstructed using the relevance-weighted sum of all the corresponding line (data) segments.

Following segment-level matching, HCMAN matches each line representation $E_V[i]$ with each column representation $E_T[m]$ using line-to-column level self-attention (LL-SAN). Similar to the segment-level process descibed above, LL-SAN uses self-attention to compute the relevance between each line and each column, and reconstructs the line chart representation E'_V and the dataset representation E'_T using the relevance-weighted sum of all the lines and columns, respectively.

In the final step, the reconstructed representation of the line chart and the dataset, E'_V and E'_T, are concatenated and then passed through an MLP to learn a relevance score $Rel'(V,T)$.

5 HANDLING DATA AGGREGATION-BASED LINE CHART QUERIES

In practical scenarios, users often generate line charts using aggregated data from datasets. For instance, daily sales figures may be aggregated to calculate total monthly sales revenue. Thus, our objective is to enhance FCM to support data aggregation (DA)-based line chart queries. Handling DA-based queries poses a greater challenge since the aggregated data used to construct the line chart can vary significantly from the original datasets. Therefore, directly applying FCM descibed above to match candidate datasets with DA-based line charts would result in poor performance due to the substantial differences between the aggregated data and the original dataset.

Moreover, the potential valid data aggregation operations for a given dataset can be extensive due to arbitrary aggregation window sizes and numerous aggregation operators. To tackle this issue, we propose an enhancement of FCM by introducing three additional layers in the dataset encoder:

- *Hierarchical Multi-scale Representation Layer (HMRL)* is proposed to capture localized characteristics associated with aggregating over a particular window size.
- *Transformation Layer* is designed to address the distribution shift between the raw candidate dataset and any aggregated data.
- *Mixture-of-Experts Layer* [55] is employed to automatically infer the most probable data aggregation operator.

5.1 Hierarchical Multi-scale Representation Layer

As introduced in Sec. 4.3, the segment-level dataset encoder learns the dataset representation in a segment-level, which is achieved by partitioning each column into several data segments. Ideally, matching the data segment length with the aggregation window size would be optimal, but this is nearly infeasible given the arbitrary nature of aggregation window sizes. To overcome this challenge, we propose the integration of segment-level representations with multi-scale information present in the columns, through the introduction of the novel hierarchical multi-scale representation layers (HMRLs). This involves further subdividing a segment w with a length of N_1 (as described in Section 4.3) into smaller segments. A tree structure is then employed to facilitate the learning of multi-scale segment representations in a bottom-up fashion. This hierarchical approach ensures that higher-level representations of segments incorporate information from their corresponding lower-level segments.

Specifically, as illustrated in Fig. 3, in HMRLs, each segment in the original dataset encoder is first divided into 2^{β} smaller segments, and then the smallest segments are organized in the leaf nodes of a binary tree. Nodes at higher levels correspond to larger segments. The j-th node in the i-th layer of the binary tree is denoted as $e_{i,j}$. For any non-leaf node $e_{i,j}$ in the binary tree, its representation can be obtained as a function of the representations of its two children, i.e., $e_{i,j} = f(left(e_{i,j}), right(e_{i,j}))$, where $right(\cdot)$ and $left(\cdot)$ denote the right child and left child of a node in the binary tree, respectively. Here, an MLP is chosen as the function f. Consequently, information at different scales is incorporated by the top node $e_{\beta,0}$. Next, we will introduce the process of obtaining the inital representation of the smallest data segment in the leaf node.

5.2 Transformation Layers

A data aggregation operator involved in a line chart can alter the distribution of the original dataset T, potentially causing a mismatch between D and T. Considering each data aggregation operator as a transformation from T to D, we propose transformation layers to capture such changes. These transformation layers model the transformation based on the representations of segments in the leaf node, facilitating the propagation of the transformation in a bottom-up manner. To achieve this, we employ a fully-connected layer with a ReLU activation function to model the transformation. Specifically, for a smallest data segment that contains a sequence of values c, the transformation layer maps it to a representation e_0 at the leaf node of the binary tree:

$$e_0 = ReLU(Wc + b), \tag{3}$$

where W and b denotes the weight matrix and the bias, respectively.

For each data aggregation operator, an independent transformation layer is applied to model the transformation, as different aggregation operators alter the original dataset in distinct ways. Additionally, an extra transformation layer is designed to model the identity transformation for Non-DA-based line charts. Consequently, there are five different transformation layers in total.

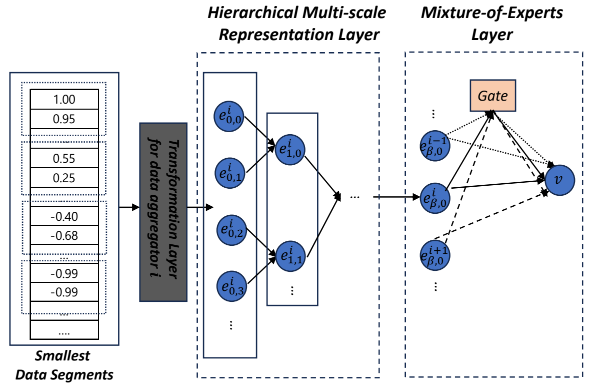

Figure 3: The architecture of the three layers for supporting aggregation-based queries.

5.3 Mixture-of-Experts Layer

We now introduce how a Mixture-of-Experts layer [45, 55] is utilized to automatically select the appropriate data aggregation operator. When dealing with a line chart V, no prior knowledge is available about whether it was generated using aggregation operators or which specific aggregation operator was applied. Despite introducing the transformation layer to bridge the data shift from the original data D to the underlying data T, the model lacks information on which transformation layer to employ. To address this issue, we introduce a mixture-of-experts (MoE) layer into FCM. MoE is widely used to combine predictions or outputs from multiple "expert" models in a weighted manner to derive a final prediction. In our context, each transformation layer is considered an expert.

For each expert model, given the five transformation layers, we obtain five different representations, denoted as $e^0_{\beta 0}, e^1_{\beta 0}, \dots e^4_{\beta 0}$. Then, the MoE layer obtains the representation v according to $v = \sum_{i=0}^{4} g_i(e^i_{\beta,0}) \cdot e^i_{\beta,0}$. Here, v is the final representation for the largest segment, which is then fed into the transformer architecture within the segment-level dataset encoder, as described in Sec. 4.3. g_i denotes the gating mechanism's output for the i-th expert. This output acts as a probability distribution over the experts, determining the weight of each expert's output. Specifically, g_i signifies the probability associated with different aggregation operators. To accomplish this, we design two fully-connected layers for each gating function g_i, using LeakyReLU and Softmax as activation functions: $g_i = \text{Softmax}(\text{LeakyReLU}(e^i_{\beta,0} W^G_1) W^G_2)$.

6 EFFICIENCY AND GENERALIZATION

In this section, we first introduce how to improve the efficiency from two aspects: offline model training and online query processing. Then, we discuss the generalization of our approach to other types of charts.

6.1 Improving Efficiency

Offline Model Training. To train the encoders and matcher in FCM, we use negative loss for the objective function:

$$\mathcal{L} = -[\frac{1}{N_{pos}} \sum_{i=1}^{K_1} r_i \cdot \log(\hat{r}_i) + \frac{1}{N_{neg}} \sum_{i=1}^{K_0} (1 - r_i) \cdot \log(1 - \hat{r}_i)] \tag{4}$$

where r_i denotes a ground truth label $Rel(D_i, T_i)$, $\hat{r}_i$ denotes the model output $Rel'(V_i, T_i)$, N_{pos} and N_{neg} denote the number of positive and negative training examples in the training dataset, respectively. Given that the original training dataset only contains positive training examples, negative training examples are created using a negative sampling strategy – for each positive training pair (V_i, T_i), we select N^- candidate datasets from the datasets to form the negative training set for V_i.

The selection of negative training examples is crucial for FCM to learn robust and discriminative representations for the dataset and line chart. Appropriate negative examples not only expedite the convergence of training processes, but also enable the model to achieve higher effectiveness. If the negative examples are too easy – markedly different from the positive examples – FCM may not effectively learn how to discriminate significant differences in the input. This can lead to under-tuned decision boundaries which fail to capture subtle nuances in the dataset. On the other hand, excessively difficult negative samples may be so close to the positive examples in the embedding space that the model fails to distinguish between them.

To ensure effective and efficient model training, we use a "semi-hard" negative example selection criteria. Without loss of generality, considering the use of an SGD-based optimization method to optimize our model in the training stage, a mini-batch B is randomly selected from the training data in each training epoch. For each line chart in B, we select semi-hard negative examples. Notably, in the training stage, access to the underlying data for all line chart queries is available. Therefore, for each line chart V, a relevance score $Rel(D, T)$ between the underlying data and all datasets in the mini-batch B is computed. Then, the datasets are ranked by the relevance score, and those with relevance scores positioned in the middle of the N^- values are included as negative examples.

Online Query Processing. Once FCM has been trained, it is ready to support dataset discovery via line charts. However, identifying the top-k relevant datasets is a computational task, and a simple linear scan will incur a high time cost. To address this issue, we employ a hybrid indexing strategy that melds interval trees with locality sensitive hashing (LSH), aimed at expediting the query process by identifying a compact set of promising candidate datasets.

Interval Tree. Recall Sec. 4.1, we can obtain the range of y-values from the line chart by the visual element extractor, which provides insights into the column range a candidate dataset should encompass. Hence, we first construct an interval tree to quickly find candidate datasets whose columns overlap with the given range–for each candidate dataset, we extract the possible range for each of its column C as an interval, defined as $[min(C), sum(C)]$, considering aggregation operations. Then, we insert all the intervals into an interval tree and employ all the intervals of a dataset to index it.

LSH. We also leverage an LSH index to expedite the query process, built upon the learned representations from FCM. First, we employ the dataset encoder introduced in Sec. 4.3 to obtain segment-level representations for each dataset. Then, for each column C of the dataset, we derive its representation E_C by averaging all representations of segments belonging to that column. Next, a hash function f is applied to map E_C to a binary code B_C. To obtain such a binary function f, we randomly generate K vectors of the same length as E_C. For each generated vector, we obtain the cosine similarity between E_C and the vector, rounding the similarity score into 0 or 1. Consequently, the binary code B_C is obtained by combining K bits of similarity scores. Note that for each dataset, it will be indexed by all binary codes of its columns.

In the query stage, given a line chart V, we first employ the visual element extractor to obtain all lines and the range of y-tick values. Treating this range as an interval, we employ it as a query on the interval tree to identify the set of datasets $\mathcal{S}_1$ that have at least one column overlapping with the query interval. Then, for the extracted lines, we employ the line chart encoder introduced in Sec. 4.2 to obtain the segment-level representations of V. Then for each line $l \in V$, we derive its representation E_l by averaging representations of segments belonging to the line. Next, the hash function f is applied to E_l to transform it into the binary code B_l. Any dataset colliding with B_l based on the binary code is added to another set $\mathcal{S}_2$. As a result, $\mathcal{S}_1 \cap \mathcal{S}_2$ contains datasets requiring further verification. For each dataset $T \in \mathcal{S}_1 \cap \mathcal{S}_2$, FCM is employed to calculate the relevance score $Rel'(V, T)$, serving as the reference for ranking. Eventually, the top-k relevant datasets are identified. The time complexity analysis can be found in the technique report [1].

6.2 Generalizing FCM

Generalization to Line Charts with numerical X-axis. In this paper, we assume that the underlying data are evenly distributed along with the x-axis since in most cases, the values on x-axis are time stamps or steps. However, there exists rare case when the x-axis values are numerical and are not evenly distributed. FCM can be fit to handle this case with two main modifications: 1) For each column C in a candidate dataset T, we assume it could be the x-axis value. Then we sort all the rows using C as a reference, and do interpolation for all the rows to make C evenly distributed. Then, FCM is able to estimate the relevance between such the derived dataset and the line chart. We denote the derived table as T'. Note that the number of such T' for T is N_C, so we need to select the one with the highest relevance score as the final score $Rel'(V, T)$; 2) Similarly, for the indexing strategy, for each T' of T, we need to add all the intervals into the interval tree to index the candidate dataset T, and add each corresponding hash code of T' to index T.

Generalization to Types of Charts. Although our focus is primarily on learning a relevance score $Rel'(V, T)$ between a line chart and a dataset, our solution can be extended to support a variety of other chart types, such as bar charts, pie charts or scatter charts, with only small adjustments: (1) Employ the visual element extractor to extract the essential visual elements from other chart types, such as sectors from a pie chart, bars from a bar chart, data point series in different colors from a scatter chart. Note that visual element extractor should be re-trained based on a related dataset. (2) Change or remove the way of segmenting the visual elements, i.e., sectors, bars, and data point series. For example, it is meaningless to further segment a bar and a sector while we can follow the same method in this paper to segment a data point series. (3) Determine an appropriate relevance metric to estimate the relevance between the underlying data and dataset, which is important to create the training dataset and identify the training examples. For instance, since a pie chart commonly depicts a data distribution, metrics such as KL-Distance may be more appropriate to compute $Rel(D, T)$.

Notably, handling certain types of charts is less challenging than line charts for two reasons: 1) The patterns in the charts may exhibit more regularity, facilitating easier representation learning by the model. 2) These charts either do not involve data aggregation (e.g., scatter charts) or have a small number of possible aggregation operations (e.g., pie charts), while line charts can use a large number of aggregation operations due to number of possible window sizes.

Table 1: Statistical properties of our benchmark (M denotes the number of lines in the line chart).

	Overall	$M = 1$	$2 \le M \le 4$	$5 \le M \le 7$	$M > 7$
Repository	19,421	9,124	4,103	3,729	2,465
Queries	400	192	86	62	58

7 EXPERIMENTS

7.1 Experimental Settings

The Benchmark. As this work is the first to explore the dataset discovery via line charts problem, no publicly available datasets exist to evaluate potential solutions. Hence, we create a benchmark dataset using data sourced from Plotly [19], a widely used dataset in data visualization [57]. Plotly contains 2.3 million records, each represented as a (table, visualization configuration) pair. The visualization configuration specifies which columns from a table and which chart type are used for the visualization. To construct the dataset for our problem, we process the Plotly data as follows, and Table 1 shows essential statistical propoerties:

- *Data Filtering and Deduplication.* Records whose datasets are not visualized using line charts are excluded, and only one record is kept if multiple near-duplicate tables exist in the records. This resulted in $13,621$ total records.
- *Table Extraction.* For each record, the table is extracted and a collection of related tables $\mathcal{D}$ is created.
- *Data Split.* The 13,621 tables are divided into a training set T_{train}, a validation set T_{val}, and a test set T_{test}. Specifically, 3,000 and 1,000 tables are randomly selected from $\mathcal{D}$ to form T_{train} and T_{val}, respectively. T_{train} is also used to create the dataset *LineChartSeg*, as described in Sec. 4.1. Furthermore, for each table in T_{train} and T_{val}, the associated visualization specification is used to create a triplet (V, D, T), as discussed in Def. 2.
- *Query Selection.* A total of 200 tables are randomly selected as T_{test} and used to construct the queries. For each query table, two types of line charts are generated: one using the corresponding visualization configuration, and the other one using data aggregation by randomly applying one of the data aggregation operators shown in Sec. 2. Specifically, we uniformly apply one of the four aggregation operators to the query table. For each table, the aggregation window size is also uniformly selected from the range $min(100, N_C/10)$, where N_C is the number of rows in the table. This results in a total of 400 line chart queries.
- *Ground-truth Generation for All The Queries.* First, for each line chart query, we inject small errors into the data for each corresponding table T of V to create 50 new tables T'. For each column $C \in T$ (excluding the column matching the x-axis), noise is added using $C_{new} = C \times \sigma$, where σ is a vector of the same length as C, and each element in σ follows a uniform distribution $U \sim (0.9, 1.1)$. These new tables T' are then added into the repository $\mathcal{T}$. In this way, we can ensure that the repository contains datasets that are similar to each line chart query. Then, for each V, the relevance score $Rel(D, T)$ (introduced in Sec. 3.1) is computed to find the top-50 tables, forming the relevant datasets.

Baselines. As this is the first study on dataset discovery via line charts, no known methods can directly solve it. Hence, we establish the following baselines for comparison:
(1) *CML.* This is a simple but effective baseline, which adopts a classic cross-modal learning framework using state-of-the-art image and table representation learning methods, a Vision Transformer [15] and TURL [7], to learn representations for line charts and datasets, respectively. Then, a matcher using cosine similarity is added to compute $Rel'(V, T)$ based on the learned representations.
(2) *Qetch*.* Qetch [36] is a sketch-based time series search method, which finds time series segments similar to a single sketched line. To extend this approach to handle queries containing multiple lines, we apply a visual element extractor to extract all of the lines from the line chart, and compute the relevance between each extracted line and each column from the candidate dataset using the matching algorithm proposed for Qetch in the original paper. Then the relevance between a dataset and a line chart obtained by aggregating all of the relevance scores between each line and each column using maximum bipartite matching, as outlined in Sec. 3.1. We denote this version of Qetch as Qetch*.
(3) *DE-LN.* We combine state-of-the-art visualization recommendation (VisRec) and visualization retrieval methods, DeepEye [33] and LineNet [34], as another baseline to solve our problem. For each candidate dataset in the repository, we apply DeepEye to generate a list of 5 candidate line charts. Then, we apply LineNet to compute the similarity between the recommended line charts and the line chart query. We use the highest similarity from all recommended line charts in the candidate dataset as the relevance score $Rel'(V, T)$.
(4) *Opt-LN.* To minimize the impact of the VisRec method accuracy, we adopt *Opt-LN* as a method to represent the *upper bound performance* of the method which combines visualization recommendation and LineNet. *Opt-LN* applies LineNet to compute the similarity between a line chart query and a candidate dataset line chart from Plotly, which is the relevance score $Rel'(V, T)$. Note that this method is not possible in practice and serves solely as an upper performance bound for DE-LN.
Evaluation Metrics. Since this is inherently a search problem, we use *prec@k* and *ndcg@k* to measure effectiveness. *prec@k* measures the accuracy of the top-k list by counting the number of correctly predicted relevant datasets in the top-k datasets returned, and *ndcg@k* measures how well a ranked list performs by positional relevance. In this work, $k = 50$, which is equal to the number of relevant datasets available for each line chart query, as described in Sec. 7.1.
Model Configuration. For the transformers used in FCM, the number of transformer encoder layers and multiple attention heads are set to 12 and 8, respectively. The dimensionality of the transformer is set to 768 by default. The line segment and column segment sizes are set to 60 and 64, respectively. The number of negative samples N^- is set to 3 by default. We use the Adam optimizer with a learning rate of 10^{-6} and train FCM for 60 epochs.
Environment. We conduct all experiments on a Ubuntu server with an Intel Core 13700K CPU (16 cores, 4.5 GHz) and a RTX 4090 GPU of 24 GB memory. Source code is available at [1].

7.2 Top-k Effectiveness

Effectiveness using All Queries. Table 2 reports the average *prec@50* and *ndcg@50* of all methods using all of the queries, with a per query breakdown with and w/o data aggregation. Our FCM achieves the best overall performance across all queries and metrics. When compared against the best baseline CML, the relative improvement is 42.3% and 54.2% in terms of *prec@50* and *ndcg@50*, respectively. CML also demonstrates the ability to locate relevant datasets, marginally outperforming Opt-LN. Note that both CML and our FCM are using a transformer-based architecture for the encoders for both line charts and datasets. This reinforces a commonly held belief in the ML community that transformers are effective on learning accurate and comprehensive representations for different data modalities. Moreover, FCM outperforms Qetch* significantly. One possible reason is that Qetch is mainly designed to match local patterns and is not suitable when matching global patterns, and the heuristic matching algorithm is not as effective as deep learning-based methods on aligning the features of line charts and datasets. Finally, Opt-LN outperforms DE-LN significantly, since the performance of DE-LN is bounded by the recommendation accuracy.

Table 2: Effectiveness for all queries and queries with/without data aggregation (DA)

	Metrics	CML	DE-LN	Opt-LN	Qetch	FCM
Overall	**prec@50**	0.149	0.091	0.121	0.112	**0.212**
	ndcg@50	0.105	0.062	0.087	0.074	**0.162**
With DA	**prec@50**	0.077	0.045	0.056	0.046	**0.187**
	ndcg@50	0.051	0.032	0.040	0.037	**0.140**
Without DA	**prec@50**	0.230	0.139	0.187	0.178	**0.275**
	ndcg@50	0.159	0.092	0.134	0.107	**0.213**

Effectiveness of Multi-line Queries. Table 3 reports the results on queries with a varying number of lines. Our FCM consistently achieves the best performance. As the number of lines M increases, the relative improvement percentage of FCM over the best baseline CML also increases. Concretely, when M falls into different ranges: 1, 2-4, 5, 6, >7, FCM surpasses CML by 27.6%, 36.4%, 48.4%, and 56.0% in terms of *prec@50*, respectively, while the *ndcg@50* of FCM surpasses CML by 39.6%, 55.6%, 69.2%, and 77.4%, respectively. The key reason is that FCM includes an effective line chart segmentation method which can extract the shape of the lines from the line chart query, and applies a finer-grained matching in the search process. This leads to superior performance for line charts that have multiple lines.

Table 3: Overall effectiveness w.r.t. varying number of lines.

		CML	DE-LN	Opt-LN	Qetch*	FCM
$M = 1$	**prec@50**	0.192	0.128	0.173	0.153	**0.245**
	ndcg@50	0.126	0.092	0.116	0.105	**0.176**
$2 \leq M \leq 4$	**prec@50**	0.159	0.075	0.105	0.123	**0.217**
	ndcg@50	0.106	0.052	0.069	0.082	**0.165**
$5 \leq M \leq 7$	**prec@50**	0.095	0.068	0.078	0.083	**0.141**
	ndcg@50	0.062	0.048	0.054	0.062	**0.110**
$M > 7$	**prec@50**	0.050	0.041	0.051	0.054	**0.078**
	ndcg@50	0.033	0.028	0.035	0.036	**0.056**

Effectiveness of DA-based Queries. FCM also demonstrates excellent performance on DA-based queries, achieving a *prec@50* of 0.187 and an *ndcg@50* of 0.140, outperforming the best baseline, CML, by 59.7% and 62.7%, respectively. Furthermore, DA-based queries appear to be much more challenging than non-DA-based queries, as evidenced by the performance drop in all the methods (see Table 2). Nevertheless, FCM is affected the least. In a further breakdown of 200 aggregation-based queries, categorized by the aggregation type: *min*, *max*, *sum*, and *mean*, along with the aggregation window size, using prec@50 is shown in Table. 4. Observe that: (1) FCM excels in handling *sum* and *mean* aggregated queries, as compared to *min* and *max* aggregation queries. This could be attributed to the transformation layer's ability to learn behaviors associated with *sum* and *mean* operations more effectively. (2) When using small aggregation window sizes (i.e., $0 - 60$), FCM exhibits stable performance. However, when the window size exceeds 60, the performance of FCM begins to degrade sharply. This may be due to the aggregation window size exceeding the dataset segment size P_2 (e.g., 64), preventing FCM from capturing the localized characteristics based on the window size. Increasing P_2 is a straightforward solution, but as discussed in Sec. 9, this leads to an overall decrease in model performance.

Table 4: Breakdown of DA-based Queries using prec@50

	Aggregation Window Size				
	$0 - 10$	$20 - 40$	$40 - 60$	$60 - 80$	$80 - 100$
min	0.164	0.157	0.168	0.132	0.127
max	0.172	0.161	0.174	0.124	0.126
sum	0.214	0.227	0.229	0.165	0.147
mean	0.226	0.213	0.224	0.176	0.162

7.3 Effectiveness of Visual Element Extraction

We perform additional experiments to verify: (1) How the proposed *LineChartSeg* from Sec. 4.1 improves the performance for the line chart segmentation task; (2) How the proposed data augmentation methods from Sec. 4.1 improve the performance of LCSeg.

To accomplish these goals, LCSeg is compared against two alternatives: (1) *SAM* [25] – a state-of-the-art pre-trained image segmentation algorithm; (2) *LCSeg** – an alternative version of LCSeg which uses common data augmentation methods, such as flipping and cropping [16] when training the model. Two additional evaluation metrics are included in this study – *Mean Average precision (MAP)* and *Accuracy*. *MAP* is widely used to evaluate image segmentation, and *Accuract* measures the percentage of lines that are accurately extracted from the line charts.

Table 5 shows the results for the line chart segmentation task. LCSeg achieves the highest scores for both *MAP* and *Accuracy*. When compared against SAM, the relative improvement is 64.1% and 120.4% on *MAP* and *Accuracy*, respectively. This highlights a significant difference between the images in the pre-training of SAM and the line charts, and underscores the necessity of a new dataset, such as *LineChartSeg*, to train more effective models for the visual element extraction. Moreover, when compared against LCSeg*, LCSeg has relative improvements of 15.7% and 22.5% on *MAP* and *Accuracy*, respectively. This study provides additional evidence that the effectiveness of our data augmentation algorithm for visual element extraction tasks outperforms conventional image segmentation algorithms.

Table 5: Effectiveness on visual element extraction

	SAM	LCSeg*	LCSeg
MAP	0.45	0.63	**0.74**
Accuracy	0.29	0.52	**0.64**

7.4 Ablation Study

7.4.1 Impact of Hierachical Cross-modal Attention Network. To verify the effectiveness of using a hierarchical cross-modal attention network (HCMAN), an alternative version of FCM is included, called FCM-HCMAN, containing the following changes: (1) For the line chart encoder, embeddings for line segments are averaged to derive a new representation for each line. Then, the representations for all the lines are averaged to generate a final representation for each line chart. (2) For the dataset encoder, embeddings for all data segments of a column are averaged to obtain a representation for each column. These column representations are then averaged to yield the final representation for a dataset. Then, the representations from all of the columns are averaged to obtain the final representation for each dataset. (3) For the matching, instead of using HCMAN, the representations from the line chart and dataset are concatenated, and the combined representations are fed into a Multilayer Perceptron (MLP) to learn a relevance score.

Table 6: Effectiveness of FCM vs FCM-HCMAN

	FCM		FCM-HCMAN	
	prec@50	**ndcg@50**	**prec@50**	**ndcg@50**
Overall	**0.212**	**0.162**	0.155	0.113
$M = 1$	**0.245**	**0.176**	0.216	0.152
$2 \leq M \leq 4$	**0.217**	**0.165**	0.182	0.134
$5 \leq M \leq 7$	**0.141**	**0.110**	0.113	0.086
$M > 7$	**0.061**	**0.056**	0.046	0.036

Table 6 compares the results for FCM and FCM-HCMAN using *prec@50* and *ndcg@50*. FCM consistently outperforms FCM-HCMAN, with an improvement of 28.3% and 30.9% on *prec@50* and *ndcg@50*, respectively. Furthermore, as M increases, the relative improvement also increases. This provides further evidence of the effectiveness of the hierarchical cross-modal attention network provides finer-grained matching between a line chart and a dataset.

Table 7: Impact of data aggregation (DA) related Layers

		Overall	**With DA**	**Without DA**
FCM	**prec@50**	**0.212**	**0.187**	**0.275**
	ndcg@50	**0.162**	**0.140**	**0.213**
FCM-DA	**prec@50**	0.180	0.082	0.278
	ndcg@50	0.134	0.054	0.214

7.4.2 The Impact of Data Aggregation (DA) Layers. To validate the value of including the three DA layers, another variant of FCM, called FCM-DA, is created, by removing these layers from the model.

Table 7 presents *prec@50* and *ndcg@50* for FCM and FCM-DA. Observe that FCM outperforms FCM-DA with a relative improvement of 18.5% when using all of the queries. For DA-based queries, FCM achieves an *prec@50* score of 0.187 and a *ndcg@50* of 0.140, outperforming FCM-DA substantially, with improvements of 120.8% and 159.2%, respectively. For non-DA-based queries, the performance of FCM is very similar to FCM-DA. This experiment demonstrates that involving the three DA layers greatly enhances the model ability to support DA-based queries, while retaining the effectiveness on non-DA-based queries.

7.5 Hyper-parameter Study

7.5.1 Impact of the Number of Negative Instances N^-. Table. 8 shows the model performance as N^- varies, using *prec@50* and *ndcg@50*. When N^- increases from 1 to 3, both *prec@50* and *ndcg@50* exhibit a steady increase, demonstrating that we need a sufficient number of negative training examples for maximum model performance. When N^- increases from 3 to 6, *prec@50* and *ndcg@50* begin to stabilize. However, as we continue to increase N^-, the model performance will eventually begin to degrade, since too many negative samples may increase the number of false negatives. Considering that more negative samples will increase training time, N^-=3 appears to achieve the best balance.

Table 8: The impact of the number of negative samples

N^-	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
prec@50	0.147	0.182	0.212	0.211	0.212	0.213	0.21	0.208
ndcg@50	0.113	0.139	0.163	0.161	0.162	0.163	0.161	0.158

7.5.2 Impact of the segment sizes P_1 and P_2. To study the impact of the segment sizes P_1 and P_2 on the model performance, various combinations of settings for P_1 and P_2 are tested, and shown in Table 9. We find that: (1) The model performance is worse when P_1 or P_2 are very large, possibly because FCM no longer captures fine-grained characteristics of the lines or the columns. (2) When both P_1 and P_2 are very small, the model performance is even worse. One possible reason is when P_1 and P_2 are too small, any local characteristics of the line or column no longer exist. For example, assume that in the extreme case, when both P_1 and P_2 are 1, a line segment degenerates to a pixel and a column segment degenerates into a single data point, and no data trend is discernible at this segment size. Thus, the model only begins to achieve the best performance when both values are moderate.

Table 9: The impact of varying the segment size of P_1 and P_2

		P_2				
		16	32	64	128	**256**
P_1	**15**	0.102	0.121	0.128	0.105	0.115
	30	0.147	0.159	0.166	0.169	0.162
	60	0.126	0.166	**0.212**	0.18	0.185
	120	0.118	0.147	0.199	0.197	0.176
	240	0.107	0.115	0.186	0.175	0.169

7.6 Efficiency Study

We also provide an experiment to show the efficiency of the proposed method. We first run all the methods to identify relevant datasets using a simple linear scan algorithm. On average, CML, LN-DE, Qetch*, and FCM require 664s, 1822s, 213s, and 705s to identify relevant datasets for a given line chart query. Qetch* is faster than CML and FCM since both CML and FCM are deep learning-based algorithms that require substantial computational resources, while Qetch* relies only on a simple heuristic matching algorithm. LN-DE is the least effective since it requires two stages to compute the relevance score – visualization recommendation and relevance score estimation. However, none of the methods are real-time as the linear scan is computationally expensive, which necessitates the indexing methods to achieve a sub-linear time cost.

Table 10 demonstrates the impact of various indexing strategies on the model performance in terms of efficiency and effectiveness. As we can see, the interval tree helps to filter out most candidates where the columns are outside of the expected range, reducing the search time to 243s. Another advantage of the interval tree is that it will not eliminate false negatives, so it can achieve the same performance as a linear scan can. In contrast, an LSH index is able to filter even more non-relevant candidates, so the query time drops to 43s. However, LSH may be overly aggressive and can exclude relevant results, and exhibits a marginal reduction in overall effectiveness. By combining both indexing strategies, the query time can be further reduced to 17s. Indexing requires 2 h 32 min and 4 h 11 min to build the interval tree and the LSH index, respectively, and the memory footprint for the two methods are 3.7GB and 5.6GB, respectively. In summary, our proposed hybrid indexing method achieves a 41x efficiency gain over a linear scan, and maintains similar performance for the task of dataset discovery using line charts.

7.6.1 Impact of Semi-hard Negative Selection. In this section, we study the effectiveness and efficiency when using the semi-hard negative selection strategy introduced in Sec. 6.1. To demonstrate this, negative training examples are selected using three alternative selection strategies, and used to train FCM, and the performance against our semi-hard selection strategy is compared.

Table 10: Comparison of different indexing strategies.

	prec@50	ndcg@50	Query time (s)
No Index	0.231	0.176	705
Interval Tree	0.231	0.176	243
LSH	0.212	0.162	43
Hybrid	0.212	0.162	17

- *Random* selects the negative training examples by randomly selecting N^- datasets from each mini-batch $\mathcal{B}$ for each line chart.
- *Hard* selects the top-N^- hardest negative training examples, i.e., the datasets with the highest relevance score $Rel(D,T)$ with the underlying data D for each line chart.
- *Easy* selects the top-N^- easiest negative training examples, i.e., the datasets with the lowest relevance score $Rel(D,T)$ for the underlying data D for each line chart.

Fig. 4 demonstrates how alternative training strategies impact the convergence of model training. Observe that: (1) FCM using a semi-hard negative selection strategy starts converging first, at epoch 26, but training FCM using random, hard, or easy negative selection strategies require 37, 42 and 47 epochs to converge, respectively. (2) The semi-hard negative sampling strategy also allows the model to achieve the best performance once FCM converges, as compared to all other selection strategies. (3) FCM using the random negative selection strategy also achieves a reasonable *prec@50* score – 0.201, which is just 10.3% less than the semi-hard strategy. However, models trained using the other two negative selection strategies fail to achieve good performance after convergance. This may result from the selected negative examples being unrepresentative or too challenging for the model when trying to differentiate between the positive and negative examples due to the variance of the similarity differences.

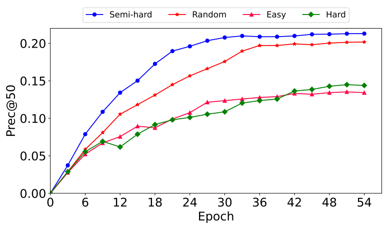

Figure 4: The impact of various negative sampling strategies on model convergence and effectiveness, using *prec@50*.

8 RELATED WORK

Dataset Discovery aims to identify relevant datasets in a large dataset repository (e.g., data lake) to meet the user need. Depending on the type of user queries, existing approaches can be broadly divided into two categories: (1) *Keywords-based dataset discovery* identify a set of datasets that are relevant to a given set of keywords; (2) *Joinable [13, 44, 58] or Unionable [2, 11, 41, 44] dataset discovery methods* find datasets that can be joined or merged with a user-provided dataset. Our work studies a novel dataset discovery problem, dataset discovery via line charts, where the input is a line chart (the query) and identifies relevant datasets which could be used to generate a line chart similar to the input.

Chart Search identifies similar charts from a repository using a chart as a query. Depending the accessibility of the underlying data, existing approaches can be divided into two categories: (1) *Data-based chart search [29, 46, 51]* assumes that the underlying data for the chart is available as input, and used to help guide the search process; (2) *Perception-based chart search [18, 31, 34, 43]* primarily study how to extract informative visual features from the charts and use similarity between features to identify similar charts. Both chart search and our work use a visual chart as the primary input. However, while chart search seeks to identify similar visualizations, the problem of dataset discovery via line charts aims to provide a collection of relevant datasets, and is considerably more complex as it requires two different data modalities to be considered.

Time Series Search. Time series uses a sequence of data points collected or recorded at regular time intervals, and is usually visualized as a line chart. A lot of work has been devoted to finding relevant time series data using different forms of user input, such as an exemplar time series [42, 53, 54], a SQL-like language [20], a regular expression [47], or a human sketch [37, 38]. Among all of the related work, sketch-based time series search is the most relevant to this work, which aims to find time series segments with a similar shape using a human sketch. However, there are two main differences between sketch-based time series search and FCM: (1) Sketch-based time series focus on localized patterns, and aim to find time series segments similar to a sketch segment, while FCM focus on global patterns, and aim to find whole datasets; (2) The proposed FCM is able to support queries with multiple lines and data aggregation, while existing sketch-based time search methods can not.

Image Segmentation is a technique widely used in computer vision to divide an image into multiple regions, each corresponding to real-world objects. Existing approaches can be broadly categorized into two types: semantic segmentation [12, 48, 56] and instance segmentation [3, 4, 32, 52]. For semantic segmentation, every pixel in the image is assigned a class, such as car, tree or building, and provides a detailed understanding of the objects in the image. Instance segmentation extends semantic segmentation by distinguishing between individual instances of the same object class. In this work, we use instance segmentation to extract visual elements from line charts. A new segmentation dataset called *LineChartSeg* is provided in this work, and a novel data augmentation method suitable for training line chart segmentation models is proposed.

9 CONCLUSION

In this work, we study a novel dataset discovery problem, dataset discovery via line charts. Given a line chart and a repository of candidate datasets, find the relevant datasets that can generate line charts which are similar to the input. To solve this new problem, we propose a fine-grained cross-modal relevance learning model (FCM), which matches a line chart and a dataset in a fine-grained manner. We also extend FCM to support DA-based line chart queries that include some types of data aggregation (DA), which is far more challenging in practice. Since this is the first study of problem, we also provide a new benchmark dataset and use it to evaluate our proposed model. The experimental results demonstrate the effectiveness of FCM to find relevant datasets for a given line chart query.

REFERENCES

[1] Anonymous. 2024. Source Code and Techinical Report. https://anonymous.4open.science/r/DDLC-0FCA/.

[2] Alex Bogatu, Alvaro AA Fernandes, Norman W Paton, and Nikolaos Konstantinou. 2020. Dataset discovery in data lakes. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 709–720.

[3] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. 2019. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9157–9166.

[4] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. 2019. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*. 9157–9166.

[5] Dan Brickley, Matthew Burgess, and Natasha Noy. 2019. Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *The World Wide Web Conference*. 1365–1375.

[6] Anh Dang, Abidalrahman Moh'd, Evangelos Milios, and Rosane Minghim. 2016. What is in a rumour: Combined visual analysis of rumour flow and user activity. In *Proceedings of the 33rd Computer Graphics International*. 17–20.

[7] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: table understanding through representation learning. *Proceedings of the VLDB Endowment* 14, 3 (2020), 307–319.

[8] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment* 1, 2 (2008), 1542–1552.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[10] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. 1994. Fast subsequence matching in time-series databases. *ACM Sigmod Record* 23, 2 (1994), 419–429.

[11] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J Miller. 2023. Semantics-Aware Dataset Discovery from Data Lakes with Contextualized Column-Based Representation Learning. *Proceedings of the VLDB Endowment* 16, 7 (2023), 1726–1739.

[12] Mingyuan Fan, Shenqi Lai, Junshi Huang, Xiaoming Wei, Zhenhua Chai, Junfeng Luo, and Xiaolin Wei. 2021. Rethinking bisenet for real-time semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9716–9725.

[13] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1001–1012.

[14] Omer Gold and Micha Sharir. 2018. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Transactions on Algorithms (TALG)* 14, 4 (2018), 1–17.

[15] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. 2022. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence* 45, 1 (2022), 87–110.

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*. 2961–2969.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[18] Enamul Hoque and Maneesh Agrawala. 2019. Searching the visual style and structure of d3 visualizations. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 1236–1245.

[19] Kevin Hu, Michiel A Bakker, Stephen Li, Tim Kraska, and César Hidalgo. 2019. Vizml: A machine learning approach to visualization recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[20] Silu Huang, Erkang Zhu, Surajit Chaudhuri, and Leonhard Spiegelberg. 2023. T-rex: Optimizing pattern search on time series. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–26.

[21] Jessica Hullman, Nicholas Diakopoulos, and Eytan Adar. 2013. Contextifier: automatic generation of annotated stock visualizations. In *Proceedings of the SIGCHI Conference on human factors in computing systems*. 2707–2716.

[22] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[23] Daniel A Keim, H-P Kriegel, and Mihael Ankerst. 1995. Recursive pattern: A technique for visualizing very large amounts of data. In *Proceedings Visualization'95*. IEEE, 279–286.

[24] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.

[25] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. 2023. Segment anything. *arXiv preprint arXiv:2304.02643* (2023).

[26] Inseok Ko and Hyejung Chang. 2017. Interactive visualization of healthcare data using tableau. *Healthcare informatics research* 23, 4 (2017), 349–354.

[27] Hans-Peter Kriegel, Marco Pötke, and Thomas Seidl. 2000. Managing intervals efficiently in object-relational databases. In *VLDB*, Vol. 20. 0.

[28] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[29] Fritz Lekschas, Brant Peterson, Daniel Haehn, Eric Ma, Nils Gehlenborg, and Hanspeter Pfister. 2020. Peax: Interactive visual pattern search in sequential data using unsupervised deep representation learning. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 167–179.

[30] Haotian Li, Yong Wang, Aoyu Wu, Huan Wei, and Huamin Qu. 2022. Structure-aware visualization retrieval. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–14.

[31] Haotian Li, Yong Wang, Aoyu Wu, Huan Wei, and Huamin Qu. 2022. Structure-aware visualization retrieval. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–14.

[32] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. 2018. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8759–8768.

[33] 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, 101–112.

[34] Yuyu Luo, Yihui Zhou, Nan Tang, Guoliang Li, Chengliang Chai, and Leixian Shen. 2023. Learned Data-aware Image Representations of Line Charts for Similarity Search. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–29.

[35] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*. 950–961.

[36] Miro Mannino and Azza Abouzied. 2018. Expressive time series querying with hand-drawn scale-free sketches. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.

[37] Miro Mannino and Azza Abouzied. 2018. Expressive time series querying with hand-drawn scale-free sketches. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.

[38] Miro Mannino and Azza Abouzied. 2018. Qetch: Time series querying with expressive sketches. In *Proceedings of the 2018 International Conference on Management of Data*. 1741–1744.

[39] Damien Masson, Sylvain Malacria, Daniel Vogel, Edward Lank, and Géry Casiez. 2023. ChartDetective: Easy and Accurate Interactive Data Extraction from Complex Vector Charts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–17.

[40] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. 2021. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence* 44, 7 (2021), 3523–3542.

[41] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825.

[42] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 262–270.

[43] Babak Saleh, Mira Dontcheva, Aaron Hertzmann, and Zhicheng Liu. 2015. Learning style similarity for searching infographics. *arXiv preprint arXiv:1505.01214* (2015).

[44] Aécio Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. 2021. Correlation sketches for approximate join-correlation queries. In *Proceedings of the 2021 International Conference on Management of Data*. 1531–1544.

[45] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2016. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*.

[46] Tarique Siddiqui, Albert Kim, John Lee, Karrie Karahalios, and Aditya Parameswaran. 2016. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. *arXiv preprint arXiv:1604.03583* (2016).

[47] Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, and Aditya Parameswaran. 2020. Shapesearch: A flexible and efficient system for shape-based exploration of trendlines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 51–65.

[48] Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. 2021. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*. 7262–7272.

[49] Steven L Tanimoto, Alon Itai, and Michael Rodeh. 1978. Some matching problems for bipartite graphs. *Journal of the ACM (JACM)* 25, 4 (1978), 517–525.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[51] Qitong Wang and Themis Palpanas. 2021. Deep learning embeddings for data series similarity search. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 1708–1716.

[52] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. 2020. Solov2: Dynamic and fast instance segmentation. *Advances in Neural information processing systems* 33 (2020), 17721–17732.

[53] Renjie Wu and Eamonn J Keogh. 2020. FastDTW is approximate and generally slower than the algorithm it approximates. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2020), 3779–3785.

[54] Chin-Chia Michael Yeh, Huiyuan Chen, Xin Dai, Yan Zheng, Junpeng Wang, Vivian Lai, Yujie Fan, Audrey Der, Zhongfang Zhuang, Liang Wang, et al. 2023. An efficient content-based time series retrieval system. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 4909–4915.

[55] Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. 2012. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems* 23, 8 (2012), 1177–1193.

[56] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. 2018. Context encoding for semantic segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 7151–7160.

[57] 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 & Data Mining*. 2389–2399.

[58] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*. 847–864.

A PRELIMINARIES

A.1 Notation Table

Key notations used are summarized in Table 11.

Table 11: A summary of key notations

Notations	Description
T	A dataset
C	A column in a dataset
V	A line chart
D	Underlying data of a line chart
d	A data series in d
$Rel(D, T)$	Relevance score between D and T
$Rel'(V, T), Rel'_{\Theta}(V, T)$	Relevance score between V and T
P_1, P_2	Number of segments in a line and a column
E_V, E_T	Representations for V and T
e	Representation for a column segment

A.2 Optional Visual Elements in a Line Chart.

Besides the mandatory visual elements, there are several optional visual elements in a line chart:

- The *Title* describes the primary topic of the line chart.
- *Labels* delineate the nature of the data represented along each axis.
- The *Legend* names each data series or individual element within the chart.

B TIME COMPLEXITY ANALYSIS

During the query stage, the time cost comes mainly from the encoding of all the candidate datasets (the time for visual element extraction and encoding the line chart query can be neglected, since they just need to be performed only once) and the matching of the line chart and each candidate dataset. Since both the encoder and matcher adopt the transformer-based architecture, the time complexity comes mainly from self-attention mechanism. Each calling for self-attention mechanism is $O(Kn^2)$, where K is the embedding size and n is the number of segments. Furthermore, it needs to invoke the self-attention mechanism once, the total encoding time for all the datasets is bounded by $O(K|\mathcal{T}|n_c(\frac{n_r}{P_2})^2)$, where n_r and n_c denotes the largest number of rows and columns in repository $\mathcal{T}$ and P_2 is the length of the data segments as introduced in Sec. 4.3. Similarly, the total matching time is bounded by $O(K|\mathcal{T}|n_c M(\frac{n_r}{P_2} + \frac{W}{P_1})^2)$, where W is the width of the line image and P_1 is the width of a line segment image as introduded in Sec. 4.2, and M denotes the number of lines in the line chart. Combining the two terms, the final time complexity is $O(K|\mathcal{T}|n_c M(\frac{n_r}{P_2} + \frac{W}{P_1})^2)$. Note that in practice, the search time could be greatly reduced since we can employ the y-ticks information to filter many irrelevant columns, as discussed in Sec. 4.3.