City Research Online

City, University of London Institutional Repository

Citation: Sun, Y., Li, J., Chen, S., Andrienko, G., Andrienko, N. & Zhang, K. (2022). A learning-based approach for efficient visualization construction. Visual Informatics, 6(1), pp. 14-25. doi: 10.1016/j.visinf.2022.01.001

This is the published version of the paper.

This version of the publication may differ from the final published version.

Permanent repository link: https://openaccess.city.ac.uk/id/eprint/28154/

Link to published version: https://doi.org/10.1016/j.visinf.2022.01.001

Copyright: City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

Reuse: Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

City Research Online: http://openaccess.city.ac.uk/ publications@city.ac.uk

Visual Informatics 6 (2022) 14–25

Contents lists available at ScienceDirect

Visual Informatics

journal homepage: www.elsevier.com/locate/visinf

A learning-based approach for efficient visualization construction

Yongjian Sun [a], Jie Li [a,*], Siming Chen [b], Gennady Andrienko [c,d], Natalia Andrienko [c,d], Kang Zhang [e]

[a] College of Intelligence and Computing, Tianjin University, China
[b] School of Data Science, Fudan University, China
[c] Fraunhofer Institute IAIS, Germany
[d] City University London, UK
[e] Beijing Normal University-Hong Kong Baptist University United International College, China

ARTICLE INFO

Article history:
Received 18 December 2021
Received in revised form 20 January 2022
Accepted 20 January 2022
Available online 31 January 2022

Keywords:
Learned index
Neural network
Visualization index
Interactive exploration
Spatiotemporal visualization

ABSTRACT

We propose an approach to underpin interactive visual exploration of large data volumes by training Learned Visualization Index (LVI). Knowing in advance the data, the aggregation functions that are used for visualization, the visual encoding, and available interactive operations for data selection, LVI allows to avoid time-consuming data retrieval and processing of raw data in response to user's interactions. Instead, LVI directly predicts aggregates of interest for the user's data selection. We demonstrate the efficiency of the proposed approach in application to two use cases of spatio-temporal data at different scales.

© 2022 The Authors. Published by Elsevier B.V. on behalf of Zhejiang University and Zhejiang University Press Co. Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Indexing is a common technique for optimizing data retrieval. By mapping data content to memory addresses, an index can quickly derive the results of a queried key by avoiding the traversal of storage space.

A state-of-the-art perspective is that an index can be regarded as an AI model. This approach is called the **Learned Index (LI)** (Kraska et al., 2018). For example, a B-Tree is a regression tree that maps a key to a position within a key-sorted set with the guarantee that the key of the record at that position is the first key equal or higher than the look-up key. Meanwhile, a bloom filter is a binary classifier that predicts whether a key exists in a set. A straightforward advantage of using these models is their fast query speed with a low storage cost. Moreover, index quality can be greatly improved, because each model is specifically trained for a dataset, thus the inherent characteristics of the dataset could be effectively captured.

Inspired by this idea, we propose the **Learned Visualization Index** (LVI), which is a neural network-based model that could be trained by collecting samples of a large number of data subsets and the corresponding visual feature values. LVI can directly output the values of the visual features subject to a data selection, thus avoiding data retrieval and visual feature computation, which are time-consuming, especially for large datasets. Using LVI#1 in Fig. 1 as an example, in which the user selects a rectangular area on a map to generate a histogram showing the number of tweets posted on each day-of-week. Having received the selected region, the LVI directly outputs the pixel heights of the seven bars without having to traverse the dataset multiple times to compute the aggregated measures.

The inherent AI features of LVI bring many distinctive advantages. First, LCI can cope with the challenge of big data by skipping the data retrieval for real-time interactive exploration. Second, LVI size depends on the relatively small-size inner structure of the neural network rather than the large volumes of original data. We therefore can apply an LVI in an environment with limited computation and storage resources. Moreover, the LVI approach supports users to query arbitrary attributes to explore different categories of data patterns. For example, we could select a spatial range to generate a visualization showing data distributions on time (Fig. 1(b-c)), explore the spatial patterns within specified time intervals (Fig. 1(a)), and conduct combined queries across space, time, and multiple attributes to achieve more comprehensive analysis goals, as in Fig. 8.

Defining, training and optimizing an LVI for a given dataset is challenging due to the following three aspects:

Information richness of visualization (C1). A visualization typically integrates multiple visual features, whereas data selection based on querying several continuous attribute intervals contains less information. It is inherently difficult for an AI model to output such rich information based on a low-dimensional

* Corresponding author.
E-mail address: jie.li@tju.edu.cn (J. Li).

https://doi.org/10.1016/j.visinf.2022.01.001
2468-502X/© 2022 The Authors. Published by Elsevier B.V. on behalf of Zhejiang University and Zhejiang University Press Co. Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

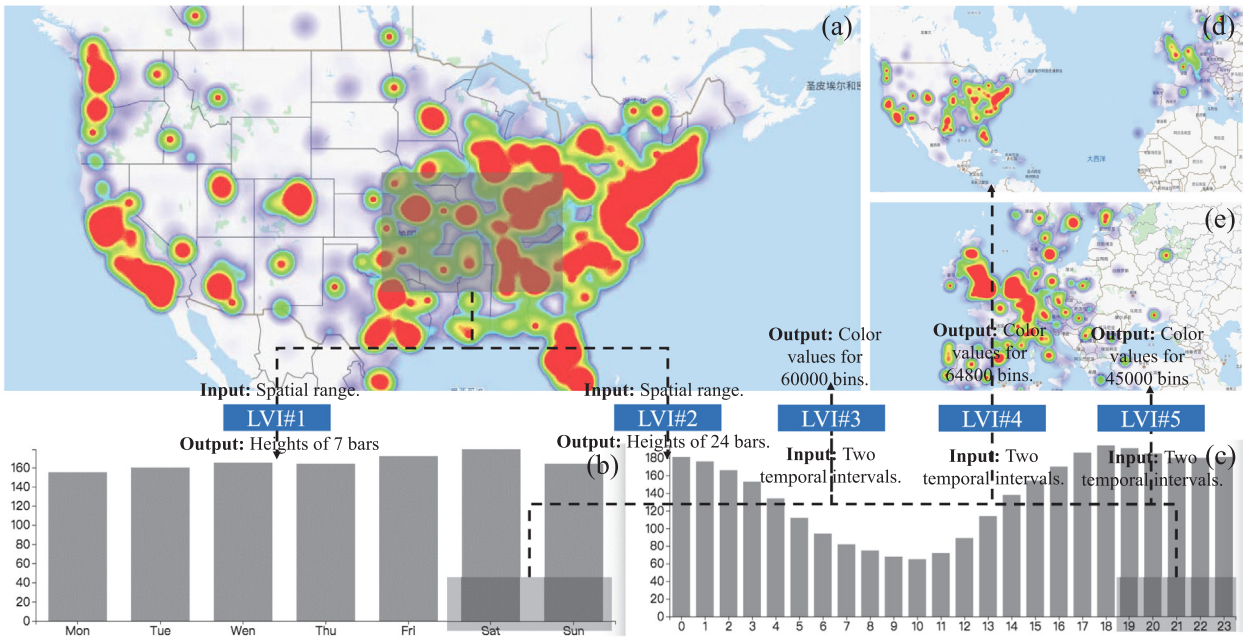

Fig. 1. Visual analysis of Brightkite tweets by using five LVIs (#1-5). Each LVI is a neural network-based model, which takes user's data selections as the input and outputs visual features for the visualization. The analyst brushes (#1-2) a rectangle on the map to view temporal patterns of the tweets posted in the selected region on days-of-week (b) and hours-of-day (c), (#3) selects two time intervals to observe spatial patterns of the posted tweets (a), and (#4-5) zooms (d) and pans (e) the map to different scales and regions.

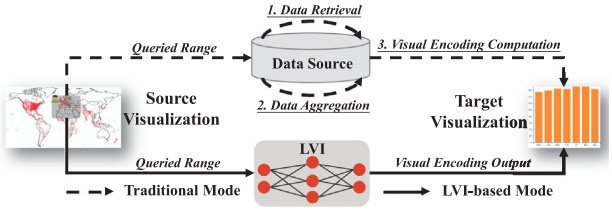

Fig. 2. Traditional (above) and LVI-based (bottom) visualization modes. Three time-consuming processes in traditional mode are marked with underline.

encoding of data selections as inputs. Ensuring fast and accurate output of values for multiple visual features poses a great challenge for LVI design.

Irregularity of data distribution (C2). Predicting the value of a visual feature can be viewed as a task of learning the distribution of the visualized measure. Such distribution may appear to as very smooth at a top-level view, but actually is irregular when zooming in to individual records, as in Fig. 3. It is well-known that the neural network is more effective to approximate the general shape of a Cumulative Data Distribution (CDF), but has a problem with being accurate at the individual data instance level (Kraska et al., 2018). Therefore, ensuring the accuracy of the predicted results is very difficult even for a single visual feature.

Huge amount of training samples (C3). Different from traditional AI models (e.g. image classifiers,) in which training sets are collected from the real world and manually labeled by a large number of humans, the training set of an LVI is generated by executing queries against the target dataset. Respectively, it is easy to get a large number of training samples by setting different query thresholds. Refining such samples for bringing the training time to an acceptable level is the key to ensure the usability of our approach.

To address these challenges, we decompose a visualization into multiple visual features, train a neural network for each visual feature separately, and design a framework that enables multiple neural networks to execute in parallel (C1). To avoid learning irregular data distributions, we train the neural network to output visual feature values rather than attribute values (C2). We leverage a Summed Area-Table (SAT) -based query sampling to reduce the number of query samples while maintaining a high accuracy (C3).

In summary, the main contributions of the paper are as follows:

- A data exploration **mode** based on AI model operations for approximating visual feature values, replacing time-consuming data retrieval. This mode enables exploration flexibility under low memory requirements and high query efficiency.
- A **strategy** to output visual feature values instead of data measures. LVI essentially learns high-level trends rather than irregular distributions of the data measures. Our approach fits the properties of neural networks, thus ensuring the accuracy of results (C2).
- An **implementation** of the LVI, which integrates an SAT-based query sampling to reduce the size of the training set (C3), a decomposition mechanism to change a visualization into multiple predictable visual feature sets (C1), and a tensor-structured framework that enables multiple separately trained neural networks to execute in parallel (C1).
- An **evaluation** on two datasets of different scales, in which multiple LVIs with different inputs and outputs are trained for analyzing their performances in different application scenarios.

The paper is structured as follows. After introducing the problem statement (Section 2), we review the related work (Section 3), present our approach (Section 4) and describe two case studies (Section 5), followed by a discussion in Section 6 and conclusions in Section 7.

2. Problem statement

Our goal is to design and implement LVI and show its utility for creating interactive visualization systems that integrate multiple views and rich interaction techniques for providing comprehensive analysis tools for finding data patterns from different perspectives.

2.1. Concept definition

To introduce LVI, we formally define the following concepts:

Definition 1. We define q as a query of visualization. Without loss of generality, q can be viewed as a set of continuous value ranges of attributes, i.e., $q = (r(a_1), r(a_2), \ldots, r(a_m))$, in which $r()$ is a range operation and $r(a_i)$ will return a continuous interval of the value range of the attribute a_i.

Definition 2. We define $Q = (q_1, q_2, \ldots, q_n)$ as a query template that consists of multiple queries on a set of attributes $A(Q) = \{a_1, a_2, \ldots, a_m\}$. From a geometric perspective, Q is a high-dimensional query space, with each dimension corresponding to an attribute of $A(Q)$. Each query q can be viewed as a subspace of Q, which conducts a range operation on a dimension (attribute). The maximum number of queries involved in Q is equal to the number of the combination of value intervals that the attributes of $A(Q)$ could take.

Definition 3. Assuming V is a visualization, without loss of generality, it can be decomposed into several visual feature sets, denoted as $V = \{VF_1, VF_2, \ldots, VF_k\}$, where $VF_i = \{vf_{i1}, vf_{i2}, \ldots, vf_{ij}\}$ is a visual feature set consisting of multiple visual features. For example, a histogram has only one set of visual features, in which a visual feature encodes the height of a bar in pixels, as in Fig. 1(b-c).

Definition 4. For a visual feature set $VF = \{vf_1, vf_2, \ldots, vf_k\}$, we use $vf(q)$ to represent the quantitative value of a visual feature $vf \in VF$ corresponding to a query q, while $VF(q)$ is a vector consisting of the values of all the visual features belonging to VF.

Definition 5. We define $e()$ as an embedding function for transforming a query q to a low-dimensional real-value vector, i.e. $e(q) \in R^m$, where m represents the length of the vector.

Definition 6. Given a query template $Q = (q_1, q_2, \ldots, q_n)$ and a visual feature set VF, we call VF can be predicated by Q if we could train a model to predict the vector of VF using the training data $D = ((e(q_1), VF(q_1)), (e(q_2), VF(q_2)), \ldots, (e(q_n), VF(q_n)))$. We call V is predictable, if for any $VF \in V$ there is a query template that can predict it, otherwise V is unpredictable.

Definition 7. Let $V = \{VF_1, VF_2, \ldots, VF_k\}$ be a predictable visualization, the LVI of V is a set of models $LVI(V) = \{f_1, f_2, \ldots, f_k\}$. Each model f_i is responsible for the prediction of a visual feature set VF_i, i.e., $f_i : Q_i \rightarrow VF_i$, where Q_i is a query template that could predict VF_i. We train models for individual visual feature sets separately because they depend on different query templates, therefore the models differ in structure and cannot be integrated together.

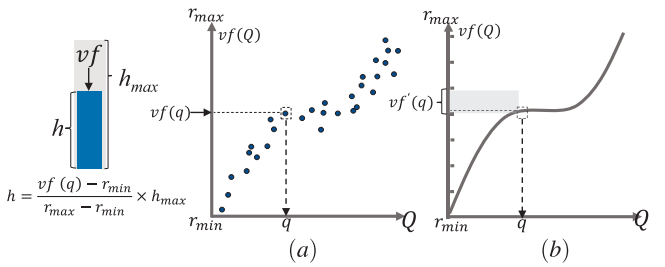

Fig. 3. Comparison of two strategies for predicting the pixel height of a bar. (a) To predict the exact measure value, which needs to learn the irregular data distribution. (b) LVI only ensures the predicted value to be correctly transformed to the pixel height, thus allowing for a certain deviation between the predicted value and the true value.

2.2. General considerations

There are several important aspects that should be discussed before designing the LVI.

The first aspect is **why we expect a neural network to be able to accurately predict values of visual features**. Let us consider a histogram with only one bar as an example. To determine the height of the bar, a straightforward idea is to train a neural network to predict the value of the visualized measure, and then use a simple linear transformation to calculate the pixel height, as in Fig. 3(a). Achieving this goal is difficult, since the distribution of the measure may be irregular, while the neural network is only suitable to learn the overall shape of the data distribution and is not good at achieving the required last mile accuracy (Kraska et al., 2018). However, for drawing the bar it is unnecessary to know exact measure values. To explain this point, let h_{max} be the maximum pixel height of the bar. This implies that the value range of the visualized measure is divided into h_{max} intervals. For calculating the height of the bar in pixels it is necessary to adjust the value to the intervals. In other words what an LVI should do is to ensure the predicted value could fall into the same interval as the actual value. This allows a certain deviation in the predicted value. We essentially need to learn a high-level smooth trend of the distribution of the measure, as in Fig. 3(b). This goal can be achieved by a neural network of a relatively simple structure.

Another important aspect to be explained is **the relationship between LVI and Learned Index LI** (Kraska et al., 2018). Although LVI is inspired by LI, they are essentially two different techniques. In particular, LI is not optimal for constructing visualizations. While LI efficiently predicts which data records satisfy a given query, further time-consuming aggregation and processing is necessary for mapping the predicted values to visual primitives. Another aspect is accuracy. As LI is a method for optimizing database retrieval, its accuracy must be strictly guaranteed, while LVI by its definition allows some deviations in the output. Respectively, LVI does not require complex structures such as the recursive structure and hybrid training used by LI for improving the accuracy. While LI processes attributes one by one for predicting data records selection, thus requiring further processing for mapping to visual features, we design LVI in a way to process multi-attribute queries at once, directly producing values of discrete visual features rather than raw values of continuous attributes. This should enable real-time efficient visualization update in response to queries on different visualizations.

3. Related work

The current trend is to optimize the visual exploration of large data volumes by applying data cubes, AI approaches, and, recently, learned index.

3.1. Data cube

Most of existing works depend on interactive exploration of large datasets relies on designing specific data structures. Data cube can be viewed as the most classic OLAP (Online Analytical Processing) (Chaudhuri and Dayal, 1997) model that allows retrieving aggregated measures on any combination of attributes in real-time. OLAP was successfully used in many domains such as social media (Cao et al., 2015; Li et al., 2018b), traffic (Shekhar et al., 2002; Chen et al., 2017), graphic analysis (Tian et al., 2008; Zhu et al., 2020; Han et al., 2021), Scatterplots analysis (Xie et al., 2021; Ma et al., 2018), etc. The main problem of the Data Cube is that the combination explosion of attribute values may result in an unacceptable storage cost seriously affecting its usability. Liu et al. (2013) propose imMens that resolves this challenge for spatio-temporal exploration by limiting the maximum number of attributes to be queried to 4. Lins et al. Lins et al. (2013) propose a quadtree-based data structure, named Nanocubes, which supports realtime spatio-temporal exploration on arbitrary ranges of space, time and attributes. Pahins et al. (2016) propose HashedCubes, which implements a similar function using a better space-efficient structure. Miranda et al. (2018) propose a structure for analyzing long time series. Mei et al. (2019) proposed an R-tree-based space partitioning scheme to enable flexible binning strategies in the exploration of tabular datasets. The core of these techniques is to design a unified data structure, in which the pre-computed aggregated measure values are stored in a hierarchical structure to ensure access efficiency. The size of the data structure is usually very large, even for simple visualizations. On the contrary, the LVI is an AI model specifically trained for a visualization, whose size is much smaller.

There exist several data cube-based techniques designed for specific purposes. Li et al. (2018a) design Semantics Space-Time Cube used to explore unstructured texts with spatial and temporal information. Wang et al. (2017) propose Gaussian Cubes for generating visualization involving complex machine learning algorithms, such as PCA and linear regression. Li et al. (2018c) propose ConcaveCubes for drawing a boundary-based cluster map. Miranda et al. (2017) propose TopKube for exploring ranking results. These approaches, however, are used to generate specific types of visualizations. Our approach is different, providing theoretically-grounded support to arbitrary types of visualizations by decomposing them into multiple predictable visual feature sets and training separate models for them.

Our LVI approach differs from the existing data cube-based techniques (Lins et al., 2013; Pahins et al., 2016) in query schemes and memory requirements. For the data cube-based techniques, hierarchical aggregation is applied to the dimensions, and the corresponding aggregated measures are pre-calculated and stored in a unified data structure according to the query scheme. The problem is two-fold. First, the query scheme is fixed, mainly because the coordinate of each measure in the data structure corresponds to the query condition, which cannot be changed after the data structure is generated. Second, the data structure always consumes a large storage space (from hundreds of MBs to dozens of GBs), depending on the size of the target dataset and the number of query conditions (coordinates in the data structure). Even for simple visualizations that only contain a few visual features, the size of the data structure cannot be reduced. Different from them, an LVI is specifically trained for a given visualization. The size of an LVI depends on its inner weight structures, which is typically very small, ranging from dozens of KBs to several MBs. The small size makes it possible to train multiple LVIs with different query schemes to implement a comprehensive visual analysis system.

3.2. AI in visualization

Combining visualizations with traditional machine learning techniques to construct a "Human-in-the-Loop" analytic pipeline has been the core goal of visual analytics (Amershi et al., 2014), with the following representative works (Sacha et al., 2018; Elassady et al., 2018; Zhao et al., 2019) and conceptual frameworks and guidance (Andrienko et al., 2018; Xia et al., 2021; Collins et al., 2018). There is a trend of using the term *AI+VIS* specifically for application of the neural network and its derivatives (e.g. Convolutional neural network CNN, Recurrent Neural Network RNN, Graph neural network GNN) in visualization.

The "black-box" character of AI models prompts the emergence of visualization techniques for revealing inner functionality of the AI models, i.e., Visualization for AI Explainability (Tzeng and Ma, 2005). A number of visualization techniques have been proposed for revealing different aspects of various kinds of AI models, such as Tree Boosting Methods (Liu et al., 2018b), Convolutional Neural Networks (Liu et al., 2017; Bilal et al., 2018), Recurrent Neural Networks (Ming et al., 2017), Generative Adversarial Networks (Kahng et al., 2019), Deep-Q Learning (Wang et al., 2019), and Deep Generative Models (Liu et al., 2018a). Two literature reviews (Choo and Liu, 2018; Hohman et al., 2018) give a comprehensive picture of the current research. These works can be called as *VIS for AI* because they essentially utilize visualization techniques for optimizing AI models. On the contrary, LVI is going to operate in *AI for VIS* mode by utilizing AI models for optimizing interactive visualization of large data volumes.

Though not formally proposed in any research paper, the concept *AI for VIS* has been considered as a fruitful direction in visualization by many domain experts. In a representative work by Kwon et al. (2018), machine learning techniques are used to achieve a quick estimation of the aesthetic metrics of large graphics, which is not practical for traditional algorithms due to a high computation complexity. This research motivated further works on combining graphics with machine learning techniques (Chen et al., 2019; Xia et al., 2020). Another important application of this category is to utilize AI models for evaluating the usability of a visualization (Sedlmair and Aupetit, 2015; Battle et al., 2018; Xia et al., 2022). A classifier can be trained on a large number of existing visualization techniques for predicting quality metrics of a visualization. These works, however, are different from our approach that is aimed at training a model for predicting values of visual features.

3.3. Learned index

Replacing traditional indexes by a trained AI model for accelerating data retrieval is a current trend in database research. In comparison to the traditional index techniques, an AI model is able to reduce greatly the storage space, while keeping the false positive rates (FPR) and false negative rates (FNR) within acceptable levels (Richter et al., 2015). Many existing techniques focus on the Locality-Sensitive Hash (LSH) (Wang et al., 2016; Guo and Li, 2015), which is essentially a classifier implementing the nearest neighbor search within a high-dimensional space.

Our LVI approach is inspired by the Learned Index proposed recently (Kraska et al., 2018), which designs a hybrid structure of AI models and traditional database index to implement three common types of database indexes: range index, point index, and existence index. In one of the follow-up papers (Kraska et al., 2019), the Learned Index was used for guiding the algorithm selection for optimizing database access methods and query plans. As discussed in Section 2.2, the differences between our approach and the Learned Index is the following: 1) LI queries a single attribute at once, while LVI supports combined query on multiple

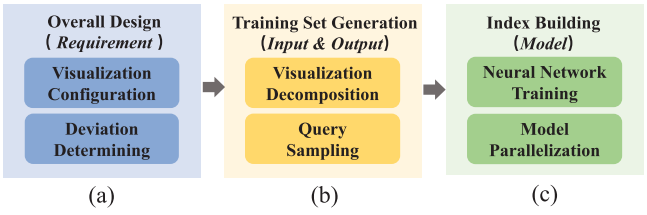

Fig. 4. The workflow of training an LVI.

attributes; 2) LI returns a list of objects, while LVI directly predicts aggregated visual feature values among the objects; 3) LI should satisfy a strict accuracy requirement, while LVI outputs a number of visual feature values with a guarantee that visual encoding is correct.

4. Our approach

In this section we introduce a general framework of our approach and provide technical details of each phase.

4.1. Workflow

Fig. 4 presents a proposed three-phase workflow of LVI training. First, we need to define a configuration of visualization (Fig. 4(a)) the LVI will be trained for. Specifically, we need to choose the type of the visualization, identify measures associated with each visual feature, and determine allowed deviations in the results.

In the second phase, the training set is generated by querying the dataset (Fig. 4(b)). We decompose the visualization into several predictable visual feature sets, each consisting of a set of visual features showing the same category of information (Definition 3). We therefore could train a model for each of them (Definition 7). Another critical step in this phase is appropriate query sampling, aimed at reducing the number of instances in the query space in the training set.

Finally, the LVI is trained using the generated training set (Fig. 4(c)). Neural networks for different visual features are trained separately and then combined in a single tensor structure, allowing them to run in parallel.

4.2. Overall design

Each LVI is trained for a specific visualization, therefore the visualization configuration should be fully determined before the model training. Specifically, to guide the training set generation we need to choose a type of visualization, assign a data measure to a visual feature, and specify how the data measure is derived from the attributes of the original dataset.

An important thing is to determine an appropriate accuracy of the target visualization. In the example in Fig. 3(b), h_{max} encoded the maximum value of the visualized measure, thus dividing the value range of the measure to a number of intervals corresponding to pixels. Depending on the number of pixels allocated to the visual feature, the desired level of LVI accuracy can be defined.

Accuracy requirements of many visualizations are not very strict, since the resolution of visual features is limited. If h_{max} of the histogram (Fig. 1(b)) is 200 pixels and the maximum number of daily tweets is about 700,000, a deviation of 3500 = 700,000/200 results in the correct height in pixels. Another example is the heatmap (Fig. 1(a)), which usually contains a limited number of colors (Harrower and Brewer, 2003), as a human eye cannot distinguish too many colors on a map (Haroz and Whitney, 2012), thus leading to more relaxing accuracy requirements. The discrete nature of visual representations enables LVI to achieve sufficient accuracy.

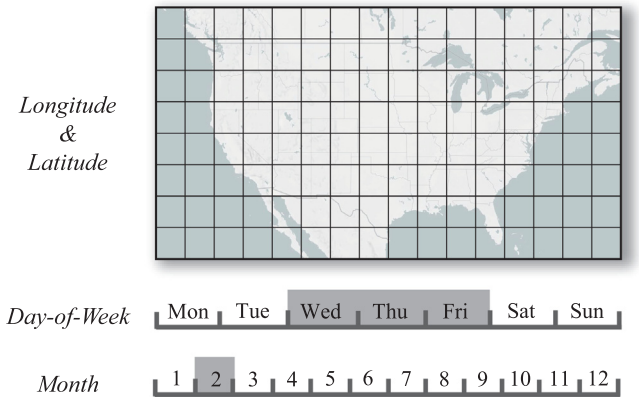

Fig. 5. Attribute discretization enables a unified representation of range queries marked as gray rectangles.

4.3. Training set generation

For training LVI, we generate the training set by executing sequences of data queries.

4.3.1. Query sampling

There is typically a large number of query instances in a query space (Definition 2). If we include all of them into the training set, it will make the training time unacceptable. We therefore utilize query sampling process to reduce the training samples with the purpose of improving the usability of LVI. The prerequisite of query sampling is to ensure an LVI to return a correct result for any query. Therefore, neither local nor random sampling is allowed, since they may exclude a part of the query space, in which the data distribution of the visualized measure is not regular enough for predicting. We therefore start with discretizing the value range of each query attribute into several bins and utilize the Summed-Area Table (SAT) -based sampling to reduce the number of training samples.

The attribute discretization is needed, first, for reducing the number of the results of the *range operation* that could be covered by a query, and, second, for providing a unified representation method for various categories of attributes, such as space, time, as well as numeric or categorical attributes. Fig. 5 shows three cases of the attribute discretization, which transforms the value range of an attribute into a small set of bins.

To improve the query granularity, the value range of an attribute is usually divided into a large number of bins, thus generating a large number of training samples. As our approach supports combined queries on multiple attributes, this problem gets even more serious. We therefore leverage Summed-Area Table (SAT) (Crow, 1984) to resolve this problem. Consider an example in Fig. 6(a), where the user trains a model for predicting the count of objects in a value range $\langle s, e\rangle$ of an attribute, denoted as $N(s, e)$. By using the SAT, we could get the desired value by subtracting the value for the range (o, S) from the value of the range (o, e), where o represents the lower boundary of the value range of the attribute: $N(s, e) = N(o, e) - N(o, s)$ (Fig. 6(a)). Similarly, we could calculate the measure of a query on two attributes using the equation $N(lb, rt) = N(o, rt) - N(o, lt) - N(o, rb) + N(o, lb)$, as in Fig. 6(b). The symbol $N()$ represents the measure value within a two-dimensional rectangle determined by a diagonal of two endpoints, and o refers to a 2D coordinate consisting of the lower boundaries of the value ranges of the two attributes. The SAT can also be used for the query on three (see Fig. 6(c)) or more attributes. The advantage of using the SAT is obvious: we need to visit each bin only once for generating the training set. In other

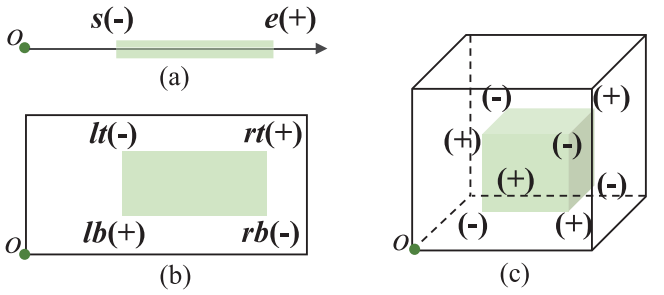

Fig. 6. SAT-based queries sampling in (a) one-, (b) two- and (c) three-dimensional query space. Each green region represents a query, while symbols + or − marked on a vertex indicates the operation conducted on the vertex for obtaining the measure value for the green region.

words, a training set only needs to include the queries whose start point is the original point o. The minimum number of the queries that should be included in a training set is $\prod_{a\in A(Q)}|a|$, where $|a|$ is the number of bins of the attribute a. This allows for keeping the number of training samples in most scenarios at an acceptable level.

For getting the measure value in an n-dimensional space using SAT, we need to conduct multiple queries in subspaces determined by o and boundary points of the target space. The measure value of the target space can be calculated through the linear combination of the results of these queries. The number of operations to get a measure value of a query template Q is $2^{|A(Q)|}$, where $|A(Q)|$ represents the number of attributes in Q. For the three cases in Fig. 6, the numbers are 2, 4 and 8, respectively. By optimizing the model structure to enable parallel execution of multiple stacked neural networks (Section 4.4.2), the time overhead of such a process is very low, not affecting the real-time update of the visualization in response to the user interaction.

It should be noted that SAT applicability is limited. It can be applied only if there are no newly-appearing and disappearing objects within the query space. Using Fig. 1 as an example, the geo-tagged tweets (objects) in the Brightkite datas are distributed throughout the world (query space), we therefore could utilize SAT to quickly get the aggregated measure of the tweets within any area of the world. As a counter-example, we cannot get the average population of a country within the time interval from 1980 to 1990 through the equation $population(1980, 1990) = population(1970, 1990) - population(1970, 1980)$, because there are newborn and dead people, causing the set of people have been changing during that period.

4.3.2. Visualization decomposition

The purpose of visualization decomposition is to transform a visualization of information richness into multiple visual feature sets with less information and more consistent form. According to Definition 3, a *predictable* visualization can be decomposed into one or multiple feature sets. For example, a histogram contains a single visual feature set, in which each visual feature encodes the height of a bar, as in Fig. 7(a). Similarly, a linechart (Fig. 7(b)) and a heatmap (Fig. 7(c)) also have one visual feature set, while a scatterplot could be decomposed into three visual feature sets that represent x and y coordinates and dot sizes, as in Fig. 7(d). It can be concluded that the number of visual feature sets is equal to the number of information categories needed for defining the visualization, regardless of the number of visual elements in the visualization. A visualization that contains tens of thousands of visual elements may have only one visual feature set, such as heatmap containing numerous colored bins (Fig. 1(a)), while a visualization with a few visual elements may have a relatively

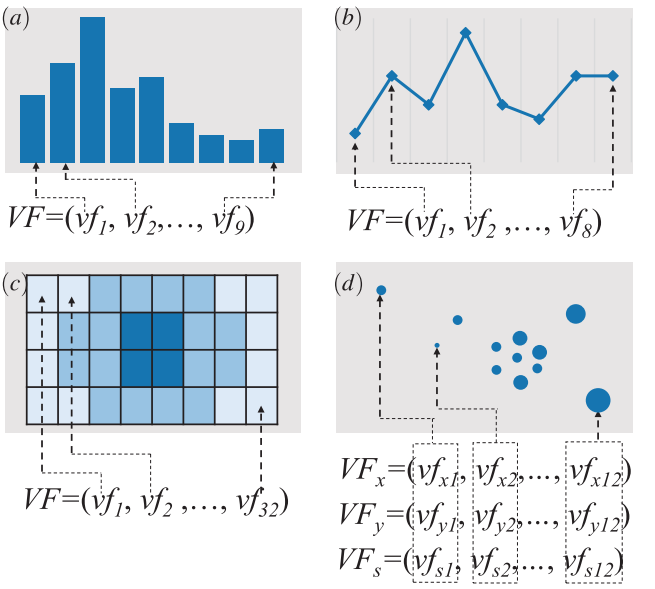

Fig. 7. Visualization decomposition of four classic visualization techniques, where (a–c) only contain one visual feature set and (d) has three visual feature sets.

more visual feature sets, such as the scatterplot in 9(c), which has three visual feature sets, but only 145 dots (visual elements). We represent each visual feature set as a vector where each value corresponds to a visual feature.

After decomposing a visualization into multiple visual feature sets, we need to define a query template for predicting them (Definition 6). A query template consists of a set of query instances on the same attributes (Definition 2), each producing a measure vector for the corresponding visual feature set (Definition 4).

Some visual features, such as object names or color schemes, do not need to be predictable (Definition 6). They are defined by the visualization environment.

4.4. Index building

Assuming VF is a visual feature set (Definition 3), the corresponding model will take an input query q and produce an output vector $VF(q)$ (Definition 4). Using Fig. 1(b) as an example, the heights of the 7 bars are independent visual features, since the tweets posted on one day could not appear on others. Training a single model for predicting multiple independent outputs is undesirable, as the model may catch occasional relationships. Instead, we propose a **Visualization Index Framework (VIF)** that enables to train a neural network for each visual feature of VF separately, as in Fig. 8(b), and then integrate weights from the separate models to a tensor structure to allow parallel execution of the neural networks (Fig. 8(c)). The purpose of the VIF is to balance accuracy and time efficiency.

Next sections describe training of the neural network for a single visual feature (Section 4.4.1) and the integration of multiple models to construct VIF (Section 4.4.2).

4.4.1. Neural network training

A neural network is responsible for predicting the value of a visual feature. The general training workflow (Fig. 8(b)) is the following:

Encoding the queries to vectors makes it easier to capture the pattern of relationships between inputs and outputs. The encoded vectors will represent the queries to be fed into the neural network. As the training set must include only the queries whose

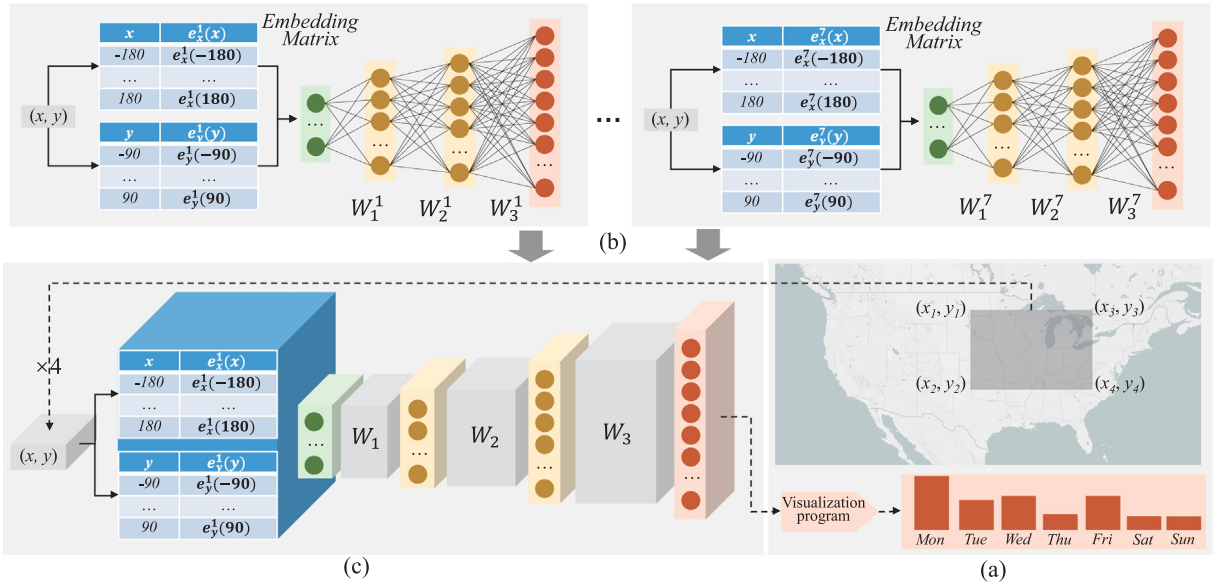

Fig. 8. Visualization Index Framework was used to construct a histogram with 7 bars. (a) The user interactively brushes a rectangle on the map to generate a histogram for the counts of objects within the rectangle on different days of week. (b) Seven neural networks, each predicting the height of a single bar. (c) A tensor model integrates 7 neural networks, executes them in parallel and outputs heights of all bars.

target ranges start from the point of origin o (Section 4.3.1), we could omit o and represent query vector as a bin coordinate in the query space. For example, coordinates (x, y) correspond to a rectangle on a map with a diagonal connecting o (the left-bottom boundary point) and (x, y) (Fig. 8(b)). We therefore only need to consider how to represent each bin coordinate as a unique vector.

We use either *one-hot* encoding or an embedding matrix to encode the query. The *one-hot* (Wang et al., 2018) encoding is the most common and effective way to encode the queries. When the attribute has fewer bins, encoding the query using *one-hot* can form a vector with fewer dimensions. For example, encoding the query from the *Day-of-Week* view in Fig. 5 will form a 7-dimensional vector: [0,0,1,1,1,0,0], because Wed, Thu and Fri were selected. But when the attributes are divided into too many bins, the one-hot encoding will lead to a long vector. This results in too many neurons being included in the input layer, thus reducing the execution efficiency of the neural network. We therefore need to represent each attribute as a low-dimensional real-valued vector. To implement this purpose, we construct an **embedding matrix** for each attribute and put them in front of the neural network, whose outputs are sequentially concatenated and fed into the neural network as the input. Using Fig. 8(b) as an example, we use two embedding matrices for the two attributes, i.e., longitude and latitude, respectively. The initial values of the embedding matrix are assigned randomly, they will be updated together with the weights of the neural network. The advantages of this structure are in two aspects. First, the structure of the embedding vector is low-dimensional, thus greatly reducing the number of necessary parameters of the neural network. Second, by connecting the embedding matrices to the neural network and updating them together, the final embedding vectors could better adapt to the calculation task of the neural network, since they reflect the predictive results to some extent. Note that there are no clear rules for choosing encoding. Typically, it is more appropriate to use embedding matrix when there are a lot of bins (e.g., more than 100) in the attributes. Instead, when the number of bins is small (e.g., less than 20), it is more effective to use *one-hot*.

The number of neurons in the input layer is determined by the length of the encoded vector of the query. A long vector could reduce the amount of the information loss, but will increase the number of parameters of the neural network. If we use embedding matrix, then it is necessary to conduct multiple trials to find the minimum length that can meet the accuracy requirement.

Neural Network Setting involves a large number of parameters that control the convergence direction of the training. Here we report only several key parameters. We utilize the simple and full-connected neural network with up to three hidden layers, typically one is enough, and the ReLU activation function to implement the model. The number of epochs is set to 1500. The training of a neural network will stop when its epoch number reaches 1500 or the prediction accuracy of the neural network reaches 100%, which will be tested within each epoch. We set the batch size to 64. The learning rate is initialized to 0.001 with a decay rate of 50% every 300 epochs. The loss function can be written as follows:

$$J = J_0 + \lambda_1 \sum_w |w| + \lambda_2 \sum_w w^2, \tag{1}$$

where λ_1 and λ_2 weight the contributions of the $L1$ and the $L2$ regularization items, which is a common method to avoid overfitting, and J_0 represents the ordinary loss function that will be set according to the implementation of the neural network.

There are two principal types of implementations of the neural network. First, we could make the neural network for predicting a single value of the visual feature, i.e. implementing the neural network as a data fitting task, in which J_0 could be set to the mean squared error (MSE). Second, we could view each value that could be taken by the visual feature as a label and implement the neural network as a multi-label classifier, in which the binary cross-entropy (BCE) can be used.

Although the second structure should involve more neural network parameters (consider there is only one neuron in the output layer of the first structure), it, however, could decrease the training difficulty. This is mainly because predicting the label of an object only needs to compare the probabilities that the object is with different labels, rather than learning the irregular data distribution. Therefore the second structure can be used in most cases, especially when there is a weak predictable relationship between the query vector and the data distribution. For simple

training tasks, such as if the visual feature allows a large deviation due to the small number of values, we could utilize the first structure for reducing the storage cost.

4.4.2. VIF for index parallelization

As the visual feature sets are independent, VIF architecture allows parallel execution of multiple neural networks (Fig. 8(c)). VIF can be regarded as a tensor-structured neural network. Given a set of neural networks with the same structure, VIF automatically extracts their weight and embedding matrices at the corresponding layer, and combines them into a tensor. This transforms multiple matrix operations into a single tensor operation of the data flowing through a layer. According to the tensor specification, operations over stacked matrices of different neural networks are conducted separately, producing the same results as what we could obtain by executing the neural networks separately.

VIF has two advantages in comparison to separately executing multiple neural networks. First, VIF only needs to record the weights and embedding metrics of the neural networks, thus avoiding the interference of unnecessary overhead and instrumentation introduced by the AI training framework, such as Tensorflow (Abadi et al., 2016) and Pytorch (Paszke et al., 2017), for managing large models (Kraska et al., 2018). Second, the compact tensor operation can be optimized more easily through the multi-thread mechanism of modern AI hardware, such as GPU or TPU, usually containing thousands of cores. Obtaining outputs through a tensor operation is theoretically slower than the data cube-based techniques that only need to access an in-cache memory address to get the pre-calculated measure. VIF, however, enables obtaining multiple values simultaneously. In other words, the time needed for getting a single value of a single visual feature is almost equal to predicting the vector for a visual feature set consisting of a number of visual features.

Note that the neural networks that could be integrated should have the same structure, i.e., the same number of neurons on each corresponding layer. Since the neural networks for predicting the visual features in a visual feature set depend on the same query template and have a similar type of outputs, it is natural to design them as having the same structure, thus making their integration possible.

Each neural network should have a standalone input. For this purpose the input vector should also be transformed into a tensor by stacking itself multiple times (each layer will be fed into a neural network), such as (x, y) coordinate of tensor structure in Fig. 8(c). This can be easily implemented through the broadcasting mechanism integrated in many deep learning frameworks.

4.5. Index training and usage

We could use any AI framework, such as Tensorflow or Pytorch, to train the neural network. After the training process, the weights and the embedding metrics of the neural network are extracted, stacked into several tensor structures, and stored in standalone files. That is, python is only used for training LVIs and we implement the tensor operations using any language with higher execution efficiency. In this paper, we implemented tensor operations using CUDA and C++. We have designed an optimizer that automatically divides a tensor operation into a large number of small matrix operations and executes each of them on a dedicated GPU core, thus greatly improving the general performance (see "Execution" parameters in Table 3).

LVI visualizations are implemented within a client–server architecture and work in a web browser. The server receives query requests from the client and sends predicted results back to the client. Our visualization component implemented in JavaScript works in the browser, maps predicted visual features to proper visual variables, handles unpredictable visual features (Definition 6) by assigning specific values to each of them, and supports interactivity such as panning and zooming, coordination of multiple views, and data selection for drilling down to patterns of interest.

Table 1
LVIs of the first case.

#	Query Template	Visualization
1	Longitude(360), Latitude(180)	Histogram(7)
2	Longitude(360), Latitude(180)	Histogram(24)
3	day-of-Week(7), hour-of-Day(24)	Heatmap(60000)
4	day-of-Week(7), hour-of-Day(24)	Heatmap(64800)
5	day-of-Week(7), hour-of-Day(24)	Heatmap(45000)

5. Evaluation

5.1. Brightkite social media checkins

The Brightkite dataset contains about 4.5M social media check-in records of 58K users collected from Apr. 2008 till Oct. 2010 worldwide. This dataset has been widely used in many papers (e.g. Lins et al. (2013), Pahins et al. (2016), Wang et al. (2018)).

We trained five LVIs to generate two histograms and three density maps (heatmaps) for different scales and regions, as in Fig. 1. Specifically, the analyst could select a spatial region to generate two histograms showing the tweets distribution on day-of-week (#1) and hour-of-day (#2), respectively. She could also select two time intervals on the two histograms to generate a heatmap for observing spatial distributions of the messages posted during the selected times (#3-5). We have trained several LVIs for supporting more efficient generation of density maps at different scales. Thus, the global map uses a global LVI (#4) with 1° resolution (360 × 180) while US (#3) and EU maps (#5) use LVIs created at finer resolutions (0.25°).

All LVIs trained for this use case are listed in Table 1. The numbers in the brackets show the number of bins for attributes in the query template and the resolution of the corresponding visualization.

5.2. Population statistics data

The second use case is based on a collection of annual statistics records of 145 countries over the period of 38 years (1971–2008) (Li et al., 2018b). The dataset contains spatial (longitudes and latitudes of countries) and temporal (year) references and three economic criteria: income per capita, life expectancy, and population count. Though this dataset is quite small, it is interesting to compare the performance of the LVI when dealing with datasets of different scales (see Fig. 2).

We train seven LVIs on this dataset to form a multiple views visualization (Fig. 9). Specifically, when the analyst selects a time point "1999" (#11), the histograms (#6-8) present distributions of values of the three attributes ("Population" in Fig. 9d, "Income per capita" in Fig. 9e, and "Life expectancy" in Fig. 9f) for all countries. A scatterplot (#9) positions the countries according to their "Income per capita" (X) and "Life expectancy" (Y), with dot sizes representing "population". In addition, an unpredictable attribute "Continent" is shown by color hues. A further selection of a spatial region on the map (#10) would support selection of countries to be shown in the scatter plot and reflected in the histograms (this operation is not illustrated). The analyst selects intervals of interest of the three attributes (#12) to see spatio-temporal aggregation of the query results. In the example, only one of the 3 attributes was affected by the query, therefore the

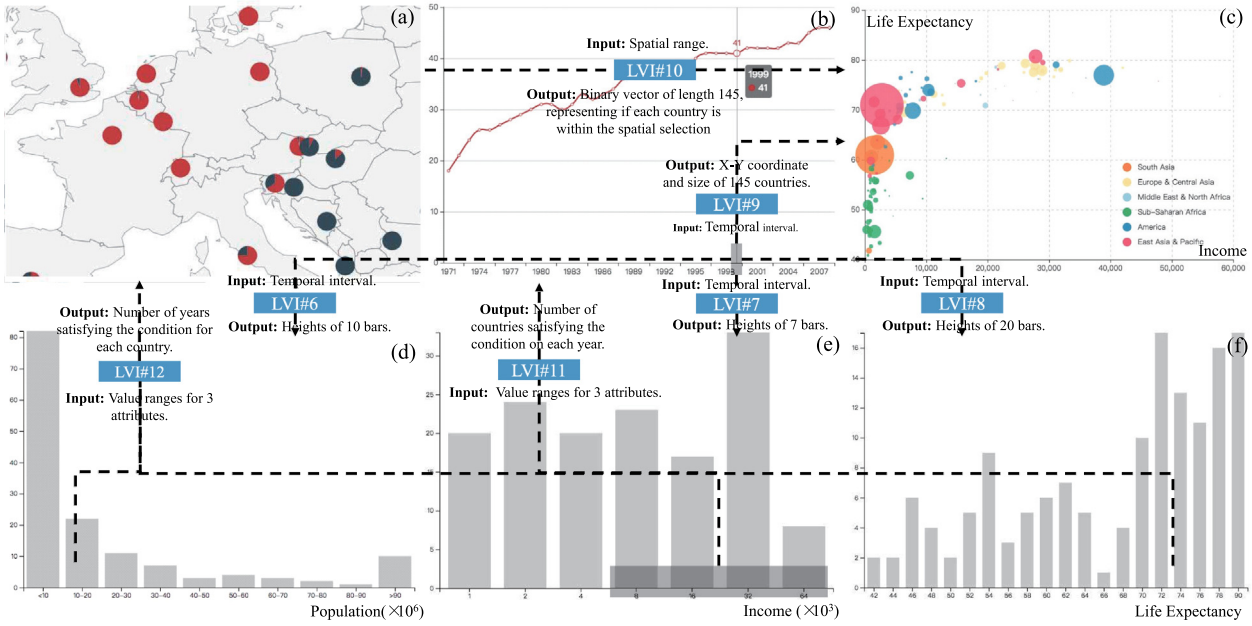

Fig. 9. Visual analysis system that integrates six views (a–f) for exploring the economic statistical data. The system is constructed based on seven LVIs (#6-12), whose inputs and outputs are marked with the corresponding visual variables (size, position, color hue and value (Bertin et al., 1983)).

Table 2
LVIs of the second case.

#	Query Template	Visualization
6	Year(38)	Histogram(10)
7	Year(38)	Histogram(7)
8	Year(38)	Histogram(20)
9	Year(38)	Scatterplot(145)
10	Longitude(360), Latitude(180)	Scatterplot(145)
11	Population(40), Life Expectancy(40), Income(40)	Linechart(38)
12	Population(40), Life Expectancy(40), Income(40)	Thememap (145)

full ranges of the remaining attributes were selected. In the result, for each country on the map (Fig. 9a) we see for how many years the conditions were satisfied. On the time graph (Fig. 9b), we see how many countries satisfy the query conditions each year.

The characteristics of all LVIs are listed in Table 2.

5.3. Performance assessment

Table 3 summarizes the model parameters and the performance criteria of the LVIs. Table 4 shows the results of the comparison of query overhead with and without LVI. All of the LVIs were trained on a GPU Server (XEON E5-2680, 128G, 1080Ti), and the execution time were tested on an ordinary PC (Core i7 6700HQ, 16G, GTX950M), thus emulating a real application environment. The execution time thus will further improve when separating client and server and using a server with a better performance.

The numbers of attributes in the query template $|A(Q)|$, training samples $|Q|$ and count of output visual features $|VF|$ are in the first three columns. Two further parameters, "Embedding" and "Weights", describe the structure of the embedding matrix and the neural network for a single neural network ("Null" means the *one-hot* encoding is used); they must be multiplied by $|VF|$ for the complete VIF structure. The remaining columns list six critical performance criteria. " Σ Size" represents the size of the files storing the embedding matrices, weights and bias of an LVI. The ratio "$|Key|/|SAT|$" representing volumes of LVI and the corresponding SAT, reflects the storage efficiency of our approach in general. We used LVI to build the two systems along with a database version on the Brightkite dataset and the Population dataset. The database version has the same interface as the LVI version. Instead of using LVI, the database version uses the database to respond user queries. The database used here is MySQL (MySQL, 2001). We randomly extract 1000 actual user queries from logs of the visualization systems and test the execution time by running them on the two versions of systems. The result is shown in Table 4. Several findings regarding the LVI performance can be summarized as follows:

(1) In general, the results support our arguments well, i.e. an LVI could accurately ($>$ 99.8%) output a large number of visual feature values (up to 64800) in real-time (3–14 ms).
(2) The "$|Key|/|SAT|$" of most LVIs are less than 50% (except for #6-9), confirming a low storage cost of our approach.
(3) If a query template includes only a single attribute (#6-9), LVI has no advantage in storage space, since the length of its embedding vector inevitably covers all bins of the attribute.
(4) The "$|Key|/|SAT|$" parameter is inversely proportional to "Deviation". For a small allowed deviation, i.e. a high accuracy requirement, the neural network (#1-2) requires more parameters. On the contrary, much fewer parameters are needed if the accuracy requirement is relaxed (#10).
(5) The size of an LVI is not correlated with the size of the original dataset. A small dataset may need a large LVI to meet complex visualization requirements (#11-12). On the contrary, to support a visualization with fewer visual features or a low query granularity, the size of the LVI could be very small, even for a large dataset (#1-2).
(6) Only one LVI cannot get 100% accuracy (#11), since no deviation is allowed in its predicted results. Such an extremely high accuracy requirement is caused by a small number of values that could be output from the LVI (integer between 0 and 145) and relative more visual feature values (0-200px), thus even a deviation of one will result in incorrect visual effects.
(7) Most of the training processes can finish within 2.5 h, which illustrates the effectiveness of the query sampling. Two LVIs (#2, 11) take longer time, because more epochs should be conducted to satisfy the accuracy requirement.

Table 3
Summarization of model parameters and performance criteria.

#	$\|A(Q)\|$ (N)	$\|Q\|$ (N)	$\|VF\|$ (N)	Embedding (X)	Weights (X)	$\|Key\|/\|SAT\|$ (%)	Σ Size (KB)	Training (h)	Execution (ms)	Deviation (N)	Accuracy (%)
1	2	64800	7	540 × 20	40 × 80,80 × 200	46.3	532	2.1	6	1/200	100
2	2	64800	24	540 × 20	40 × 80,80 × 200	46.3	1822	6.0	7	1/200	100
3	2	168	60000	Null	31 × 1	18.5	7495	2.1	13	1/5	100
4	2	168	64800	Null	31 × 1	18.5	7495	2.3	14	1/5	100
5	2	168	45000	Null	31 × 1	18.5	5621	1.8	11	1/5	100
6–8	1	38	20	Null	38 × 1	100	3	0.6	3	1/200	100
9	1	38	145	Null	38 × 3	100	65	1.5	3	1/200,1/50	100
10	2	64800	145	540 × 5	10 × 2	4.2	1541	0.4	2	1/2	100
11	3	64000	38	120 × 20	60 × 80,80 × 146	29.5	3727	14.4	7	1/146	99.8
12	3	64000	145	120 × 20	60 × 60,60 × 39	13	2481	1.2	5	1/39	100

Table 4
Comparison of execution time between LVI version and database version.

		LVI	Database
Brightkite	Mean	8.6ms	8.5s
	Max	15.8ms	35.4s
	Std	1.3ms	7.3s
	Median	8.4ms	7.9s
Population	Mean	3.1ms	0.3s
	Max	9.0ms	1.2s
	Std	0.8ms	0.2s
	Median	3.1ms	0.3s

(8) In real scenarios, the maximum execution time of LVI does not exceed 20 ms. LVI always executes fast no matter how many records are in the dataset. Compared to the database version, LVI can reduce more query overhead on large datasets.

6. Discussion

In this paper, we propose a neural network-based approach to constructing interactive visualization systems with coordinated multiple views. The resulting client–server system is able to handle large and complex data in efficient way, enabling fast update of visualizations in response to interactively specified queries. The proposed approach can be used for implementing a flexible OLAP, free from pre-defined intervals on cube dimensions.

Due to the inherent features of the neural networks, our approach enables real-time response to queries on arbitrary attribute ranges with higher accuracy at a low storage cost. However, we also acknowledge several limitations.

Generality. LVI cannot predict unpredictable visual feature set that contains non-numerical visual features. However, if the visual features in the unpredictable visual feature set are constant values, we can just store these values and visualize them directly. For visualizations that do not contain unpredictable visual feature set, no matter how complex the visualization is, we can decompose them into a set of numerical vectors for prediction. Complex visualization may contain more visual feature sets, but does not increase the complexity of the training process.

Storage Space. Although the size of an LVI is typically small, especially in comparison to the original data, it is more appropriate to benchmark it against the corresponding SAT that delivers precise results. Essentially, the purpose of LVI is to learn the distribution of the SAT. Depending on the regularity of the data distribution and allowed deviation, the parameter amount of the LVI may take from 1/20 to 1/2 of the SAT size.

Accuracy. The desired accuracy of an LVI is based on the accuracy requirements, i.e. allowed deviation of the target visualization. Naturally, more parameters and more complex structures should be used for achieving higher accuracy. For complex query templates consisting of numerous attributes, the predictable relationship between the query template and the visual feature set may become weak. However, the embedding matrix can effectively alleviate this problem. Another issue is overfitting. Our method is used for interactive exploration. Fitting all training samples is enough, which includes all possible queries. This is quite different from existing deep learning models that inevitably meet new inputs in application domains.

Parallelization. The VIF approach conceptually allows multiple neural networks to be executed in parallel by splitting integral tensor operations to a large number of pieces and executing them separately on different hardware cores. At present, it is common that a GPU contains thousands of cuda cores, theoretically supporting the same number of tasks to be executed in parallel. However, this is not strictly necessary, as visualizations usually have only tens to hundreds of visual features, which is far less than the number of cores of a common AI hardware. Several visualizations containing more visual features, such as heatmap ($\#3-5$), which could also be calculated within several computation cycles.

Flexibility. A distinguishable characteristic of our approach is that each LVI is specifically trained for a selected visualization, and the visual design should be fixed before the training process. Although this limitation restricts flexibility of use, it can greatly save storage space by avoiding a huge data structure, in which the theoretical amount of information is not fully utilized, especially for generating simple visualizations.

Update. The cost for updating an LVI is significantly higher than for a traditional database index, such as B-Tree, and data cube-based techniques, which only need to update all nodes on the branch of the hierarchical index structure from the root to the leaf node where the change occurs. This, however, is a common problem that all AI applications are facing. We expect that the development of modern AI hardware will increase the training speed.

Conceptually, the problem of updates can be resolved for data that adds over time, which is a common setting for dashboard applications. The time range can be divided into bins (e.g. of one hour), and neural networks are trained separately for each time bin. For producing outputs for the whole time range, the outputs of networks for separate time bins are combined. Such an approach limits the applicability to additive aggregate measures (sum, count) and their derivatives (average), but does not allow such aggregates as median or count distinct identities. The time overhead for training multiple networks is compensated by their smaller dimensionality. So, there exists a class of dashboard applications that can be supported with dynamic data updates.

Scalability. There are two aspects of scalability: the data amount and the query resolution (the number of bins included in the query template). The data volume affects the training time, but does not affect the storage space of the final LVI. Increasing of the query resolution causes a longer encoding, thus introducing

more parameters to the neural network. However, query embedding that reduces query dimensionality could greatly alleviate this problem.

Functionality. Our approach is suitable for visualizations that represent additive aggregation measures such as sum, count, and average. Visualizations of other kinds (e.g. representing positional measures such as medians or quantiles, or complex derived measures such as low-dimensional embeddings of multi-dimensional data) are not supported. The query primitives are restricted to *range operations*, but are not able to support multiple ranges selection or more complex interaction techniques such as Lasso query. However, the lasso polygon can be decomposed into a set rectangles of different size and shape, and results of the corresponding queries are combined.

Despite these limitations, our approach has a good potential to be applied in a large number of practically important applications that require interactive visualization of large data volumes. It works especially good if the desired visualization accuracy is not very high (e.g. in apps that are supposed to be used on mobile devices).

7. Conclusion and future work

This paper has presented Learned Visualization Index (LVI), which is a neural network-based technique that supports efficient interactive exploration of large datasets. LVI is a machine learning technique that can achieve the same functions as traditional data structure-based IDE techniques. We have shown that LVI has state-of-the-art performance and advantages absent in existing techniques. Particularly, the storage cost of LVI can beat all existing techniques by not generating a complex data structure. The easy process of developing the prototype IDE systems reflects better usability. We thus believe LVI can replace traditional techniques in specific cases. As a novel AI4VIS application, LVI brings a new direction for future IDE research.

In the future, we plan to make two improvements. First, we plan to apply it to further datasets for a thorough evaluation of its performance. Second, the LVI can be viewed as a general concept of predicting values of visual features instead of computing them from the data. We are going to experiment with other AI techniques for extending our approach (e.g. CNN, RNN, and GNN), and consider possibilities to support more complex visualizations such as trees and graphs and more flexible interactions such as lasso query.

CRediT authorship contribution statement

Yongjian Sun: Conducts the main body of the work and experiments. **Jie Li:** Proposes the idea and write the manuscript. **Siming Chen:** Help in refining the idea and paper organization. **Gennady Andrienko:** Help in refining the idea and paper organization. **Natalia Andrienko:** Help in refining the idea and paper organization. **Kang Zhang:** Responsible for the final paper proofing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical Approval

This study does not contain any studies with human or animal subjects performed by any of the authors.

Acknowledgments

This work is supported by National Key R&D Program of China (2018YFC0831700), NSFC project (61972278), Natural Science Foundation of Tianjin (20JCQNJC01620), and the Browser Project (CEIEC-2020-ZM02-0132).

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al., 2016. Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design And Implementation ({OSDI} 16). pp. 265–283.

Amershi, S., Cakmak, M., Knox, W.B., Kulesza, T., 2014. Power to the people: The role of humans in interactive machine learning. Ai Mag. 35 (4), 105–120.

Andrienko, N.V., Lammarsch, T., Andrienko, G.L., Fuchs, G., Keim, D.A., Miksch, S., Rind, A., 2018. Viewing visual analytics as model building. Comput. Graph. Forum 37 (6), 275–299.

Battle, L., Duan, P., Miranda, Z., Mukusheva, D., Chang, R., Stonebraker, M., 2018. Beagle: Automated extraction and interpretation of visualizations from the web. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. ACM, p. 594.

Bertin, J., Berg, W.J., Wainer, H., 1983. Semiology of Graphics: Diagrams, Networks, Maps, Vol. 1. University of Wisconsin press Madison.

Bilal, A., Jourabloo, A., Ye, M., Liu, X., Ren, L., 2018. Do convolutional neural networks learn class hierarchy? IEEE Trans. Vis. Comput. Graphics 24 (1), 152–162.

Cao, G., Wang, S., Hwang, M., Padmanabhan, A., Zhang, Z., Soltani, K., 2015. A scalable framework for spatiotemporal analysis of location-based social media data. Comput. Environ. Urban Syst. 51, 70–82.

Chaudhuri, S., Dayal, U., 1997. An overview of data warehousing and OLAP technology. SIGMOD Rec. (ISSN: 0163-5808) 26 (1), 65–74.

Chen, W., Guo, F., Han, D., Pan, J., Nie, X., Xia, J., Zhang, X., 2019. Structure-based suggestive exploration: A new approach for effective exploration of large networks. IEEE Trans. Vis. Comput. Graphics 25 (1), 555–565.

Chen, W., Huang, Z., Wu, F., Zhu, M., Guan, H., Maciejewski, R., 2017. VAUD: A visual analysis approach for exploring spatio-temporal urban data. IEEE Trans. Vis. Comput. Graphics 24 (9), 2636–2648.

Choo, J., Liu, S., 2018. Visual analytics for explainable deep learning. IEEE Comput. Graph. Appl. 38 (4), 84–92.

Collins, C., Andrienko, N.V., Schreck, T., Yang, J., Choo, J., Engelke, U., Jena, A., Dwyer, T., 2018. Guidance in the human–machine analytics process. Vis. Inf. 2 (3), 166–180.

Crow, F.C., 1984. Summed-area tables for texture mapping. In: SIGGRAPH.

Elassady, M., Sevastjanova, R., Sperrle, F., Keim, D.A., Collins, C., 2018. Progressive learning of topic modeling parameters: A visual analytics framework. IEEE Trans. Vis. Comput. Graphics 24 (1), 382–391.

Guo, J., Li, J., 2015. CNN based hashing for image retrieval. Comput. Vis. Pattern Recog..

Han, D., Pan, J., Zhao, X., Chen, W., 2021. Netv. js: A web-based library for high-efficiency visualization of large-scale graphs and networks. Vis. Inf. 5 (1), 61–66.

Haroz, S., Whitney, D., 2012. How capacity limits of attention influence information visualization effectiveness. IEEE Trans. Vis. Comput. Graphics 18 (12), 2402–2410.

Harrower, M., Brewer, C.A., 2003. Colorbrewer.org: An online tool for selecting colour schemes for maps. Cartogr. J. 40 (1), 27–37.

Hohman, F.M., Kahng, M., Pienta, R., Chau, D.H., 2018. Visual analytics in deep learning: An interrogative survey for the next frontiers. IEEE Trans. Vis. Comput. Graphics.

Kahng, M., Thorat, N., Chau, D.H.P., Viegas, F., Wattenberg, M., 2019. GAN lab: Understanding complex deep generative models using interactive visual experimentation. IEEE Trans. Vis. Comput. Graphics 25 (1), 310–320.

Kraska, T., Alizadeh, M., Beutel, A., Chi, E., Ding, J., Kristo, A., Leclerc, G., Madden, S., Mao, H., Nathan, V., 2019. Sagedb: A learned database system. CIDR.

Kraska, T., Beutel, A., Chi, E.H., Dean, J., Polyzotis, N., 2018. The case for learned index structures. In: Proceedings of the 2018 International Conference on Management of Data. pp. 489–504.

Kwon, O.-H., Crnovrsanin, T., Ma, K.-L., 2018. What would a graph look like in this layout? a machine learning approach to large graph visualization. IEEE Trans. Vis. Comput. Graphics 24 (1), 478–488.

Li, J., Chen, S., Chen, W., Andrienko, G., Andrienko, N., 2018a. Semantics-space-time cube. a conceptual framework for systematic analysis of texts in space and time. IEEE Trans. Vis. Comput. Graphics.

Li, J., Chen, S., Zhang, K., Andrienko, G., Andrienko, N., 2018b. COPE: Interactive exploration of co-occurrence patterns in spatial time series. IEEE Trans. Vis. Comput. Graphics.

Li, M., Choudhury, F.M., Bao, Z., Samet, H., Sellis, T., 2018c. ConcaveCubes: Supporting cluster-based geographical visualization in large data scale. Comput. Graph. Forum 37 (3), 217–228.
Lins, L., Klosowski, J.T., Scheidegger, C., 2013. Nanocubes for real-time exploration of spatiotemporal datasets. IEEE Trans. Vis. Comput. Graphics 19 (12), 2456.
Liu, Z., Jiang, B., Heer, J., 2013. imMens : real-time visual querying of big data. Eurographics 32, 421–430.
Liu, M., Shi, J., Cao, K., Zhu, J., Liu, S., 2018a. Analyzing the training processes of deep generative models. IEEE Trans. Vis. Comput. Graphics 24 (1), 77–87.
Liu, M., Shi, J., Li, Z., Li, C., Zhu, J., Liu, S., 2017. Towards better analysis of deep convolutional neural networks. IEEE Trans. Vis. Comput. Graphics 23 (1), 91–100.
Liu, S., Xiao, J., Liu, J., Wang, X., Wu, J., Zhu, J., 2018b. Visual diagnosis of tree boosting methods. IEEE Trans. Vis. Comput. Graphics (1), 1.
Ma, Y., Tung, A.K., Wang, W., Gao, X., Pan, Z., Chen, W., 2018. Scatternet: A deep subjective similarity model for visual analysis of scatterplots. IEEE Trans. Vis. Comput. Graphics 26 (3), 1562–1576.
Mei, H., Chen, W., Wei, Y., Hu, Y., Zhou, S., Lin, B., Zhao, Y., Xia, J., 2019. Rsatree: Distribution-aware data representation of large-scale tabular datasets for flexible visual query. IEEE Trans. Vis. Comput. Graphics 26 (1), 1161–1171.
Ming, Y., Cao, S., Zhang, R., Li, Z., Chen, Y., Song, Y., Qu, H., 2017. Understanding hidden memories of recurrent neural networks. In: IEEE Conference on Visual Analytics Science And Technology. VAST, pp. 13–24.
Miranda, F., Lage, M., Doraiswamy, H., Mydlarz, C., Salamon, J., Lockerman, Y., Freire, J., Silva, C.T., 2018. Time lattice: A data structure for the interactive visual analysis of large time series. Comput. Graph. Forum 37 (3), 23–35.
Miranda, F., Lins, L.D., Klosowski, J.T., Silva, C.T., 2017. TopKube: A rank-aware data cube for real-time exploration of spatiotemporal data. IEEE Trans. Vis. Comput. Graphics 24 (3), 1394–1407.
MySQL, A., 2001. Mysql.
Pahins, C.A., Stephens, S.A., Scheidegger, C., Comba, J.L., 2016. Hashedcubes: Simple, low memory, real-time visual exploration of big data. IEEE Trans. Vis. Comput. Graphics 23 (1), 671–680.
Paszke, A., Gross, S., Chintala, S., Chanan, G., 2017. Pytorch. Comput. Softw. Vers. 0.3 1.
Richter, S., Alvarez, V., Dittrich, J., 2015. A seven-dimensional analysis of hashing methods and its implications on query processing. 9, (3), pp. 96–107,
Sacha, D., Kraus, M., Bernard, J., Behrisch, M., Schreck, T., Asano, Y., Keim, D.A., 2018. SOMFlow: Guided exploratory cluster analysis with self-organizing maps and analytic provenance. IEEE Trans. Vis. Comput. Graphics 24 (1), 120–130.
Sedlmair, M., Aupetit, M., 2015. Data-driven evaluation of visual quality measures. Comput. Graphics Forum 34 (3), 201–210.
Shekhar, S., Lu, C., Liu, R., Zhou, C., 2002. CubeView: a system for traffic data visualization. pp. 674–678.
Tian, Y., Hankins, R.A., Patel, J.M., 2008. Efficient aggregation for graph summarization. pp. 567–580.
Tzeng, F.-Y., Ma, K.-L., 2005. Opening the black box-data driven visualization of neural networks. In: Visualization. 2005. VIS 05. IEEE, IEEE, pp. 383–390.
Wang, Z., Cashman, D., Li, M., Li, J., Berger, M., Levine, J.A., Chang, R., Scheidegger, C.E., 2018. NNCubes: Learned structures for visual data exploration. CoRR abs/1808.08983 arXiv:1808.08983.
Wang, Z., Ferreira, N., Wei, Y., Bhaskar, A.S., Scheidegger, C.E., 2017. Gaussian cubes: Real-time modeling for visual exploration of large multidimensional datasets. IEEE Trans. Vis. Comput. Graphics 23 (1), 681–690.
Wang, J., Gou, L., Shen, H.-W., Yang, H., 2019. DQNViz: A visual analytics approach to understand deep Q-networks. IEEE Trans. Vis. Comput. Graphics 25 (1), 288–298.
Wang, J., Liu, W., Kumar, S., Chang, S., 2016. Learning to hash for indexing big data—A survey. Learning 104 (1), 34–57.
Xia, J., Chen, T., Zhang, L., Chen, W., Chen, Y., Zhang, X., Xie, C., Schreck, T., 2020. SMAP: A joint dimensionality reduction scheme for secure multi-party visualization. In: 2020 IEEE Conference on Visual Analytics Science And Technology. VAST, pp. 107–118. http://dx.doi.org/10.1109/VAST50239.2020.00015.
Xia, J., Lin, W., Jiang, G., Wang, Y., Chen, W., Schreck, T., 2021. Visual clustering factors in scatterplots. IEEE Comput. Graph. Appl. 41 (5), 79–89. http://dx.doi.org/10.1109/MCG.2021.3098804.
Xia, J., Zhang, Y., Song, J., Chen, Y., Wang, Y., Liu, S., 2022. Revisiting dimensionality reduction techniques for visual cluster analysis: An empirical study. IEEE Trans. Vis. Comput. Graphics 28 (1), 529–539. http://dx.doi.org/10.1109/TVCG.2021.3114694.
Xie, P., Tao, W., Li, J., Huang, W., Chen, S., 2021. Exploring multi-dimensional data via subset embedding. arXiv preprint arXiv:2104.11867.
Zhao, Y., Luo, F., Chen, M., Wang, Y., Xia, J., Zhou, F., Wang, Y., Chen, Y., Chen, W., 2019. Evaluating multi-dimensional visualizations for understanding fuzzy clusters. IEEE Trans. Vis. Comput. Graphics 25 (1), 12–21.
Zhu, M., Chen, W., Hu, Y., Hou, Y., Liu, L., Zhang, K., 2020. DRGraph: An efficient graph layout algorithm for large-scale graphs by dimensionality reduction. IEEE Trans. Vis. Comput. Graphics 27 (2), 1666–1676.