Learned Data-aware Image Representations of Line Charts for Similarity Search

YUYU LUO, Tsinghua University, China
YIHUI ZHOU, Tsinghua University, China
NAN TANG, QCRI, Qatar / HKUST (GZ), China
GUOLIANG LI*, Tsinghua University, China
CHENGLIANG CHAI, Beijing Institute of Technology, China
LEIXIAN SHEN, Tsinghua University, China

Finding line-chart images similar to a given line-chart image query is a common task in data exploration and image query systems, *e.g.,* finding similar trends in stock markets or medical Electroencephalography images. The state-of-the-art approaches consider either data-level similarity (when the underlying data is present) or image-level similarity (when the underlying data is absent).

In this paper, we study the scenario that during query time, only line-chart images are available. Our goal is to train a neural network that can turn these line-chart images into representations that are aware of the data used to generate these line charts, so as to learn *better* representations. Our key idea is that we can collect both data and line-chart images to learn such a neural network (at training step), while during query (or inference) time, we support the case that only line-chart images are provided. To this end, we present LineNet, a Vision Transformer-based Triplet Autoencoder model to learn data-aware image representations of line charts for similarity search. We design a novel pseudo labels selection mechanism to guide LineNet to capture both data-aware and image-level similarity of line charts. We further propose a diversified training samples selection strategy to optimize the learning process and improve the performance. We conduct both quantitative evaluation and case studies, showing that LineNet significantly outperforms the state-of-the-art methods for searching similar line-chart images.

CCS Concepts: • **Information systems** → *Database utilities and tools*; *Data analytics*; *Similarity measures*; *Top-k retrieval in databases*; • **Human-centered computing** → **Visualization toolkits**; *Visual analytics.*

Additional Key Words and Phrases: line charts; similarity search; learned representations; triplets selection

ACM Reference Format:
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. *Proc. ACM Manag. Data* 1, 1, Article 88 (May 2023), 29 pages. https://doi.org/10.1145/3588942

*Guoliang Li is the corresponding author (liguoliang@tsinghua.edu.cn).

Authors' addresses: Yuyu Luo, Tsinghua University, China, luoyy18@mails.tsinghua.edu.cn; Yihui Zhou, Tsinghua University, China, zhouyh19@mails.tsinghua.edu.cn; Nan Tang, QCRI, Qatar / HKUST (GZ), China, ntang@hbku.edu.qa; Guoliang Li*, Tsinghua University, China, liguoliang@tsinghua.edu.cn; Chengliang Chai, Beijing Institute of Technology, China, ccl@bit.edu.cn; Leixian Shen, Tsinghua University, China, slx20@mails.tsinghua.edu.cn.

2836-6573/2023/5-ART88
https://doi.org/10.1145/3588942

1 INTRODUCTION

Line charts have become a popular means to present and communicate data insights in many types of documents, news websites, and many other applications [8, 11, 19, 40, 50–59, 66, 71, 72, 75, 82]. For example, New York Times uses line charts to show the newly reported COVID-19 cases.

With the increasing usage of line chart, ***how to search similar line charts*** has attracted extensive attention from academia and industry [30, 42, 43, 60, 61, 64, 69, 73, 74]. Specifically, it can support various downstream applications, such as visualization recommendation [53, 54, 63, 79], visual query systems [40, 60, 73], visualization corpus construction [31, 83], computational journalism [15], web mining [16, 22, 40], and so on.

Given a line chart query and a repository of line charts to be queried, the line charts similarity search problem aims to retrieve a subset of the repository where each line chart is "similar" to the query. The key premise of this problem is *how to measure the similarity* between line charts. Existing studies mainly focus on quantifying the similarity between line charts based on the availability of (meta) data associated with line charts. More specifically, a line chart **L** may be associated with the underlying data **D** (*i.e.,* data for rendering) and a rendered visualization **V** in the form of an image (*e.g.,* a PNG file), as shown in Figure 1(a). Accordingly, existing approaches typically measure the similarity between line charts either based on the underlying data **D** to capture *data-level* similarity [28, 40, 42, 60, 61, 68, 73, 74] or using the rendered line chart image **V** to capture *image-level* similarity, also referred to as *visualization-level* similarity [30, 43, 64, 69].

Data-level Similarity Search of Line Charts. As shown in Figure 1(b), previous studies [28, 40, 60, 61, 68, 73, 74] utilize distance metrics (*e.g.,* Euclidean distance [21] and Dynamic Time Warping [17]) to quantify the similarity between line charts based on the underlying data **D**. Alternatively, deep learning based methods can encode **D** of line charts into learned representations such that similarity search can be conducted on these representations. For example, Peax [42] uses a Convolutional Autoencoder to learn the representations from **D** for searching similar line charts in the embedding space (see Table 1).

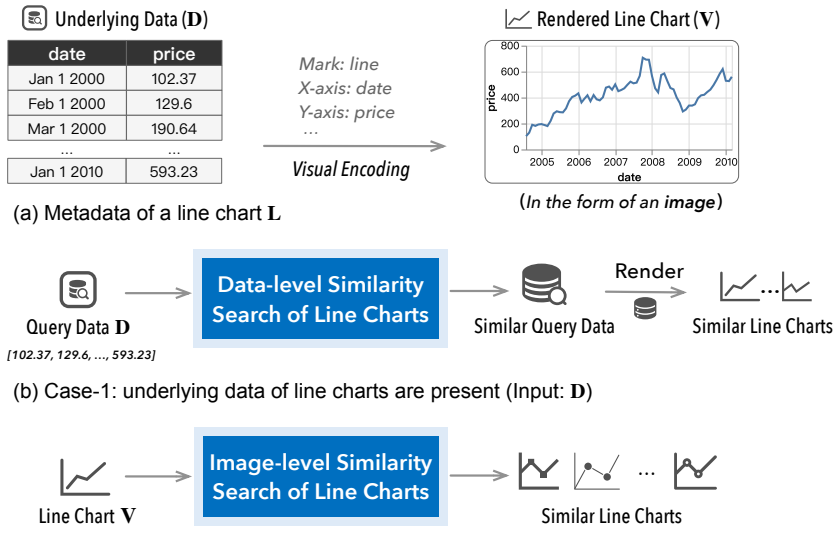


Fig. 1. Two cases of similarity search of line charts.

Table 1. Comparison of LineNet and existing approaches.

	Learned?	Available Inputs				Similarity Measure
		Training Time		*Querying Time*		
		D	**V**	**D**	**V**	
Zenvisage [73]	No	/	/	Given	Not Given	Data-level
Qetch [61]	No	/	/	Given	Not Given	Data-level
PEAX [42]	Yes	Given	Not Given	Given	Not Given	Data-level
V-CNN [43]	Yes	Not Given	Given	Not Given	Given	Image-level
LineNet (ours)	Yes	Given	Given	Not Given	Given	Data-aware Image-level

When the underlying data **D** of line charts are available *at query time*, the above approaches that measure the *data-level* similarity between line charts are simple yet effective. Unfortunately, the underlying data of line charts are not always available in many cases [43], such as searching for similar line charts in digital newspapers or when the line chart is rendered as a static image.

Image-level Similarity Search of Line Charts. In order to alleviate the problem that the underlying data of line charts are absent *at query time*, recent researches [30, 43, 64, 69] propose to directly measure the similarity between line charts by considering the visualization features of the rendered line chart image **V**, *i.e.,* the user-perceived visualization similarity as a proxy for the data similarity between line charts. As shown in Figure 1(c), generally speaking, these works [30, 43, 64, 69] take as input line chart visualizations (*e.g.,* a PNG image), then convert the visualizations into representations by leveraging deep learning models such as CNN [39], ResNet [27], and based on which finally compute the perceptual similarity to return top-k similar line chart images.

Image-level similarity search methods can handle the cases where the underlying data **D** of line charts are unavailable at query time (see Table 1). However, existing solutions rely solely on low-level visual features and may be disturbed by some visualization elements (*e.g.,* markers, legends, and structures) that do not accurately reflect the similarity between line charts. *One opportunity is to incorporate rich metadata information of line charts (*e.g., *underlying data) to better generate the image-level representations of line charts.*

Learned Data-aware Image Representations of Line Charts. To overcome the limitations of the two approaches mentioned above and achieve the best of both worlds, our goal is to train a model that can accurately *distinguish* the data distribution (*i.e.,* the characteristics of lines) of line chart images using only the rendered images **V** *at query time*. To achieve this, we propose to calibrate the model during training time with both the *underlying data* **D** and the *rendered images* **V** of line charts. This approach enables us to capture data-aware and image-level similarity more accurately, even when we *only* have rendered images **V** at query time (as shown in Table 1).

Challenges. Learning data-aware image representations of line charts faces three main challenges. **(C1)** Given the rendered images **V** and the underlying data **D** of line charts, how can we design a model that can *simultaneously* learn *better* representations by capturing both inherent image-level characteristics and the data distribution of line chart images? **(C2)** There is currently no *off-the-peg* corpus containing a large number of line charts and associated underlying data and rendered images to drive the model training and evaluate the performance. **(C3)** How can we judiciously select a set of representative training samples to effectively and efficiently train the model?

Our Methodology. Our basic idea is to *capture image-level characteristics* based on visual features of line chart image **V** by leveraging the autoencoder architecture, and to calibrate the model to *preserve the data-level similarity* based on the underlying data **D** by leveraging the deep metric learning technique. Thus, we devise a Triplet Autoencoder architecture, by taking the Vision

Transformer [48] as the backbone, to build LineNet. LineNet will be trained in conjunction with the reconstruction loss (for image-level characteristics) and the triplet loss (for data-level similarity) simultaneously (addressing **C1**). To facilitate the research on similarity search of line chart images, we build a large-scale line chart corpus, LineBench, containing over $115K$ line charts along with associated underlying data and rendered images. We also crowdsource a set of similar line chart labels that can be used for evaluation (addressing **C2**). We design a pseudo labels selection mechanism that can *automatically* generate effective *training samples* from LineBench. We further exploit a novel *diversified* training samples selection algorithm that carefully selects a set of discriminative samples to improve the training efficiency and get better performance (addressing **C3**).

Contributions. We make several notable contributions:

(1) Problem Statement. We formally define the problem of learned data-aware image representations of line charts for similarity search (Section 2).

(2) LineNet. We propose LineNet, a novel Vision Transformer-based Triplet Autoencoder model to learn *data-aware image representations of line charts* by leveraging both the underlying data and rendered images of line charts (Sections 3 & 4). We further design a diversified training samples selection algorithm to make the learning process more effective (Section 5).

(3) LineBench. We develop a large-scale line chart corpus, namely LineBench, which has more than $115K$ line charts and the metadata from four real-world datasets. We further produce a set of similarity labels between line chart images by crowdsourcing (Section 6).

(4) Evaluation. We conduct both quantitative evaluations and case studies to demonstrate that LineNet can work well both in benchmark datasets and real-world cases (Section 7).

The code and dataset for LineNet and LineBench are available at *https://github.com/Thanksyy/LineNet-and-LineBench-SIGMOD2023*.

2 PROBLEM FORMULATION

Line Chart. We consider a line chart $\mathbf{L}$ as visualization type that visualizes a list of points $(p_1, p_2, ..., p_n)$, where each point $p_i = (x_i, y_i)$, $(i \in [1, n])$, is associated with a x-value x_i and a y-value y_i.

A line chart $\mathbf{L}$ may be associated with the following metadata:

(1) $\mathbf{D}$ is the underlying data (*i.e.,* data for rendering) of $\mathbf{L}$ (*e.g.,* in Fig. 1(a)). Typically, $\mathbf{D}$ is a sequence of data points $(p_1, p_2, ..., p_n)$;
(2) $\mathbf{V}$ is the rendered visualization of $\mathbf{L}$ in the form of an image (*e.g.,* in Figure 1(a)). *We will interchangeably use the image and visualization to mention $\mathbf{V}$ if the context is clear.*

Line Chart Similarity. The key to similarity search of line charts is the distance function $\mathbf{dist}(\mathbf{L}_i, \mathbf{L}_j)$ between two line charts $\mathbf{L}_i$ and $\mathbf{L}_j$, where $\mathbf{dist}(\cdot, \cdot)$ can be Dynamic Time Warping (DTW) [17], Euclidean Distance (ED) [21], or other tailor-made distance functions. The smaller the $\mathbf{dist}(\mathbf{L}_i, \mathbf{L}_j)$ is, the more similar they are. Given two line charts $\mathbf{L}_i$ and $\mathbf{L}_j$, a distance function $\mathbf{dist}(\cdot, \cdot)$, and a threshold τ, we say that $\mathbf{L}_i$ and $\mathbf{L}_j$ are *similar* iff $\mathbf{dist}(\mathbf{L}_i, \mathbf{L}_j) \leq \tau$. In this paper, we take DTW, which is well known to measure the distance between two data series [60, 67, 73], as the default setting to compute $\mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)$.

Given a set $\mathbb{L}$ of line charts $\{\mathbf{L}_1, \mathbf{L}_2, ..., \mathbf{L}_n\}$ and a line chart query $\mathbf{L}_q$, if the underlying data $\mathbb{D} = \{\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_n\}$ and $\mathbf{D}_q$ are available *at query time*, $\mathbf{dist}(\mathbf{L}_i, \mathbf{L}_j)$ can be computed based on these data points, *i.e.,* $\mathbf{dist}(\mathbf{L}_i, \mathbf{L}_j) = \mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)$. However, if the underlying data $\mathbb{D} = \{\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_n\}$ and $\mathbf{D}_q$ are absent *at query time*, that is, *only* the rendered images $\mathbb{V} = \{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_n\}$ and $\mathbf{V}_q$ of line charts are given *at query time*, the research problem we ask is that how can we learn a representation function $\mathcal{F}(\cdot)$ of line chart images based on the underlying data $\mathbf{D}$ and image $\mathbf{V}$

during the training step, **such that the distance between $\mathcal{F}(\mathbf{V}_i)$ and $\mathcal{F}(\mathbf{V}_j)$ in the embedding space can reflect the data similarity while also capturing image-level similarity between line charts $\mathbf{L}_i$ and $\mathbf{L}_j$.** More specifically, we aim at minimizing $|\mathbf{dist}(\mathcal{F}(\mathbf{V}_i), \mathcal{F}(\mathbf{V}_j)) - \mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)|$. In this paper, we use squared Euclidean distance to compute $\mathbf{dist}(\mathcal{F}(\mathbf{V}_i), \mathcal{F}(V_j))$, which is a widely adopted measure for computing distance between vector representations in the embedding space.

Next, we formally introduce our studied problem.

Learned Data-aware Image Representations of Line Charts. Given a set $\mathbb{L}$ of line charts $\{\mathbf{L}_1, \mathbf{L}_2, ..., \mathbf{L}_n\}$, and the associated underlying data $\mathbb{D} = \{\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_n\}$ and rendered images $\mathbb{V} = \{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_n\}$, the problem of learned data-aware image representations of line charts aims to learn a neural-network based embedding function $\mathcal{F}(\cdot)$ to map every line chart image $\mathbf{V}_i \in \mathbb{V}$ to a low-dimensional length-h embedding vector $\mathbf{E}_i$ (*i.e.,* $\mathbf{E}_i = \mathcal{F}(\mathbf{V}_i)$), such that for any pair of line chart images $\mathbf{V}_i$ and $\mathbf{V}_j$, $\mathbf{dist}(\mathcal{F}(\mathbf{V}_i), \mathcal{F}(\mathbf{V}_j))$ is close to $\mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)$:

$$\arg\min_{\theta} |\mathbf{dist}(\mathcal{F}(\mathbf{V}_i), \mathcal{F}(\mathbf{V}_j)) - \mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)| \tag{1}$$

where θ denotes the neural network parameters, *i.e.,* the learned representation function $\mathcal{F}(\cdot)$ is parameterized by θ.

Therefore, our goal is to train a neural network that maps the rendered line chart images (*e.g.,* $\mathbf{V}_i$ and $\mathbf{V}_j$) into learned representations (*e.g.,* vectors $\mathbf{E}_i$ and $\mathbf{E}_j$) and the learned representations should preserve the data similarity while also capturing the image similarity, denoted by $\mathbf{dist}(\mathbf{E}_i, \mathbf{E}_j) \approx \mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)$.

If the data-aware line chart image representations function $\mathcal{F}(\cdot)$ is ready, we can support similarity search (e.g., threshold-based or top-k based) of line chart images.

3 AN OVERVIEW OF LINENET

3.1 Design and Train LineNet

Key Idea. To learn visualization representations of line charts, we need enough labeled similar/dissimilar images. However, *human-annotated examples* are far from being enough. To this end, we propose to generate "*similar/dissimilar*" pseudo labels *as a proxy for* the similarity of the corresponding line chart images $\mathbf{V}$ by computing $\mathbf{dist}(\cdot, \cdot)$ based on their underlying data $\mathbf{D}$. We then design a Vision Transformer-based Triplet Autoencoder framework, *i.e.,* LineNet. The key idea behind the framework is that we can learn the inherent image-level similarity by leveraging the Vision Transformer-based Autoencoder while capturing the data-level similarity based on the Triplet Network with the deep metric learning technique.

For training Triplet Autoencoder-based framework, each training sample is a triplet consisting of an anchor line chart image, a similar and a dissimilar one to the anchor (see Figure 2(a)-(2)).

LineNet Overview. As shown in Figure 2(a)-(3), LineNet is built using a Triplet Autoencoder architecture, which consists of three identical autoencoders. For each autoencoder, it consists of an Encoder and a symmetric Decoder. At the training phase, LineNet takes as input a line chart image triplet $(\mathbf{V}, \mathbf{V}^+, \mathbf{V}^-)$, which is comprised of an anchor, a positive (*i.e.,* similar), and a negative (*i.e.,* dissimilar) line chart image. We compute $\mathbf{dist}(\cdot, \cdot)$ based on their underlying data $\mathbf{D}$ to generate triplets, which indicates their *data similarity*. We will discuss how to select triplets in Section 4.3.

For learning data-aware image representations of line charts, we first use line chart image triplets to enforce LineNet to capture the "data similarity" between line chart images by leveraging the deep metric learning techniques [29, 70]. For example, as shown in Figure 3, the optimization objective of metric learning is to minimize the distance between the anchor (*i.e.,* blue point) and its positive samples (*i.e.,* green stars) while maximizing the distance between the anchor and negative samples

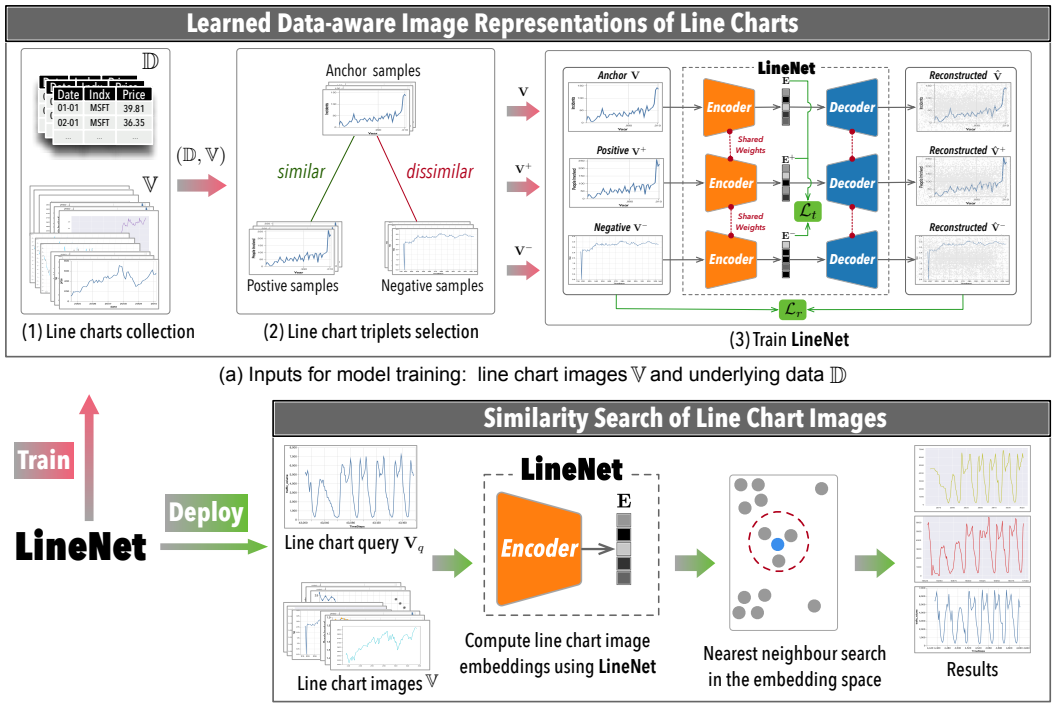


Fig. 2. The solution overview of LineNet.

(*i.e.,* pink diamonds). More specifically, given a line chart triplet $(\mathbf{V}, \mathbf{V}^+, \mathbf{V}^-)$, the optimization goal is to minimize the distance between similar line chart images and maximize the distance between dissimilar line chart images in the embedding space. In addition, LineNet needs to capture the inherent visual features of the line chart itself, which benefits the data-level similarity learning while capturing the image-level similarity. This goal is achieved by LineNet's autoencoder architecture. Specifically, the Encoder of LineNet first maps a H-dimensional input line chart image $\mathbf{V}$ into an h-dimensional compact vector representations $\mathbf{E} = \text{Encoder}(\mathbf{V})$ ($\mathbf{E} \in \mathbb{R}^h$), where $h < H$; the Decoder then reverses this procedure by reconstructing $\hat{\mathbf{V}} = \text{Decoder}(\mathbf{E})$. This learning fashion has been empirically verified in many domains that embeddings can learn useful latent information [9].

Train LineNet. We first propose a triplets selection algorithm to *automatically* select a set of training samples (*i.e.,* a set of triplets). This algorithm can produce a larger number of effective training samples for training LineNet. We discuss the details in Section 4.3. Furthermore, we judiciously select a small set of most representative triplets to make the learning process more effective and efficient. Therefore, we further design a *diversified* triplets selection algorithm to pick a set of discriminative triplets from a mini-batch to optimize the learning process. We discuss the details in Section 5.

3.2 Deploy LineNet

Suppose we have a well-trained LineNet that can map a line chart image $\mathbf{V}$ to an embedding vector $\mathbf{E}$ in the embedding space. Naturally, we can use the squared Euclidean distance between $\mathbf{E}_i$ and $\mathbf{E}_j$ in the embedding space to compute the similarity between line charts $\mathbf{V}_i$ and $\mathbf{V}_j$, which is defined as: $\mathbf{dist}(\mathcal{F}(\mathbf{V}_i), \mathcal{F}(\mathbf{V}_j)) = \mathbf{dist}(\mathbf{E}_i, \mathbf{E}_j) = ||\mathbf{E}_i - \mathbf{E}_j||_2^2$.

Therefore, once the embedding space is well-prepared, the top-k similarity search of line chart images can be tackled using k nearest neighbor search algorithms [7] in the embedding space.

As shown in Figure 2(b), given a line chart image query, and a collection of line chart images to be searched, LineNet's Encoder first maps all line chart images into a learned embedding space. Next, it employs the k nearest neighbor search algorithm to directly derive the top-k most similar results to the query in the embedding space.

Generalize to Other Visualization Types. Although we take the line charts as the exemplar to investigate the problem of learned data-aware visualization representations for similarity search, our framework can be extended to support other visualization types such as bar charts, pie charts, and scatter plots. The key to this extension is measuring data-level similarity between visualization charts based on their underlying data.

Discussion. In our attempts to capture the similarity between line chart images, we tried two intuitive solutions. The first solution is to map the representations of a pair of line chart images into a certain similarity score (*i.e.,* $\mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)$) by leveraging a regressor. However, this approach is hard to train for convergence and leads to low performance. The main reason is that line charts always have large and complicated feature space, which is rather hard to map the features to a similarity score accurately. The second one is to maximize the similarity between two representations of a pair of line chart images by minimizing a contrastive loss function. While this method can optimize the representations between similar line charts, it is challenging to handle the overall learned representations across similar and dissimilar line chart, which still results in poor performance. Please refer to Section 7 for our empirical findings regarding the effectiveness of these solutions

4 THE DESIGN DETAILS OF LINENET

We will first present the architecture of LineNet (Section 4.1). We will then elaborate on the design details of LineNet (Section 4.2). Finally, we introduce how to select training data (Section 4.3).

4.1 LineNet Architecture

To capture the visual features of line charts, we utilize the autoencoder architecture, a popular architecture for embedding learning [9, 80], to learn the inherent visual information of line chart images. We then preserve the data similarity between line chart images with the help of triplet labels, by leveraging the deep metric learning technique [29, 70].

LineNet Architecture. We design a Vision Transformer-based Triplet Autoencoder architecture to build LineNet (see Figure 2(a)-(3)). LineNet comprises *three identical autoencoders* by sharing the same weights. We adapt the sophisticated Swin Transformer block [48] as the basic unit to build the Encoder and the symmetric Decoder. We will show the design details of LineNet in Section 4.2.

For training, LineNet takes as input a set of line chart triplets $\mathbb{V}_t = \{(\mathbf{V}_1, \mathbf{V}_1^+, \mathbf{V}_1^-), (\mathbf{V}_2, \mathbf{V}_2^+, \mathbf{V}_2^-), ..., (\mathbf{V}_n, \mathbf{V}_n^+, \mathbf{V}_n^-)\}$. For a triplet $(\mathbf{V}_i, \mathbf{V}_i^+, \mathbf{V}_i^-)$, Encoder first encodes them into h-dimensional embeddings $\mathbf{E}$, $\mathbf{E}^+$, and $\mathbf{E}^-$, respectively. Next, the Decoder reconstructs origin inputs from $\mathbf{E}$, $\mathbf{E}^+$, and $\mathbf{E}^-$ and outputs the reconstructed results $(\hat{\mathbf{V}}_i, \hat{\mathbf{V}}_i^+, \hat{\mathbf{V}}_i^-)$.

Next, we will introduce how LineNet optimizes the above two goals simultaneously.

Learn Inherent Image Similarity. The autoencoder in LineNet is responsible for learning the most salient visual features of line chart images during training, which can be achieved by the reconstruction loss [45] naturally.

$$\mathcal{L}_r = \frac{1}{n}\sum_{i=1}^{n}(||\mathbf{V}_i - \hat{\mathbf{V}}_i||_2^2 + ||\mathbf{V}_i^+ - \hat{\mathbf{V}}_i^+||_2^2 + ||\mathbf{V}_i^- - \hat{\mathbf{V}}_i^-||_2^2) \tag{2}$$

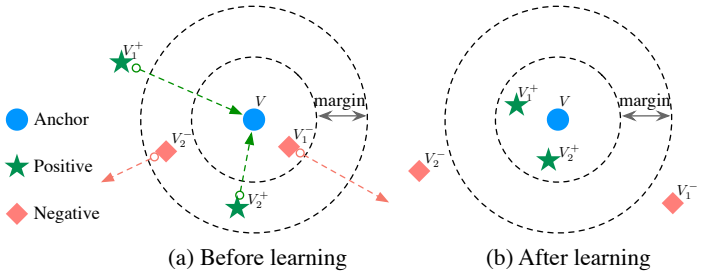


Fig. 3. An example of triplet selection for learning.

The intuition behind reconstruction loss $\mathcal{L}_r$ is to reduce the reconstruction error with the constraint of having only limited degrees of freedom available. Therefore, it enforces LineNet to retain the most representative visual features of the line chart images, which helps to capture the intrinsic image similarity and at the same time facilitates the learning process of data similarity.

Capture Data Similarity. Although the reconstruction loss can *prioritize* retaining the most representative features of the line chart images, but it has little ability to *explicitly* optimize the learning process of capturing data similarity among line chart images. Therefore, we adopt the Triplet Network to tackle this problem.

As shown in Figure 3, the Triplet Network architecture [29, 70, 78] aims at minimizing the embedding distance between the anchor (*e.g.,* $\mathbf{V}$) and its positive (*i.e.,* similar line charts) ones (*e.g.,* $\mathbf{V}_1^+$) while maximizing the embedding distance between the anchor and the negative (*i.e.,* dissimilar line charts) ones (*e.g.,* $\mathbf{V}_1^-$). This objective can be achieved by the triplet loss [78], which ensures that the negative sample is farther from the anchor than the positive sample by at least a margin. The triplet loss $\mathcal{L}_t$ for preserving data similarity between line charts is written as below:

$$\mathcal{L}_t = \frac{1}{n}\sum_{i=1}^{n} max\{0,\ \beta + ||\mathbf{E}_i - \mathbf{E}_i^+||_2^2 - ||\mathbf{E}_i - \mathbf{E}_i^-||_2^2\} \tag{3}$$

The triplet loss function $\mathcal{L}_t$ indicates that the distance of an anchor-positive pair $||\mathbf{E}_i - \mathbf{E}_i^+||_2^2$ should be smaller than an anchor-negative pair $||\mathbf{E}_i - \mathbf{E}_i^-||_2^2$ by a certain margin β.

Example 1. *As shown in Figure 3(a), Green arrows indicate reducing the distance to the anchor (blue point) to update the embedding (Fig. 3(b) shows the results), while pink arrows indicate increasing the distance to the anchor. The triplet* $(\mathbf{V}, \mathbf{V}_1^+, \mathbf{V}_1^-)$ *is useful for metric learning because the positive sample* $\mathbf{V}_1^+$ (i.e., *the similar ones) is far from the anchor* $\mathbf{V}$ *while the negative* $\mathbf{V}_1^-$ *is close to the anchor, which violates the margin significantly and leads a large error. Therefore, the* $\mathbf{V}_1^+$ *will be pushed close to the anchor* $\mathbf{V}$ *and the* $\mathbf{V}_1^-$ *will be pulled away from* $\mathbf{V}$ *(see Fig. 3(b)) due to the deep metric learning.*

Overall Objective and Optimization. According to the above definitions, the final objective function of LineNet can be formulated by a linear combination:

$$\mathcal{L} = (1-\alpha)\mathcal{L}_r + \alpha\mathcal{L}_t \tag{4}$$

where α is a hyper-parameter that balances the contribution of each loss term. The LineNet will be trained in conjunction with the reconstruction loss and the triplet loss simultaneously.

4.2 Details of LineNet: Vision Transformer

Inspired by the recent advancements of vision transformers in the computer vision community [20, 24, 34, 48], we build LineNet based on Swin Transformer [48], which has shown superior performance on vision tasks recently.

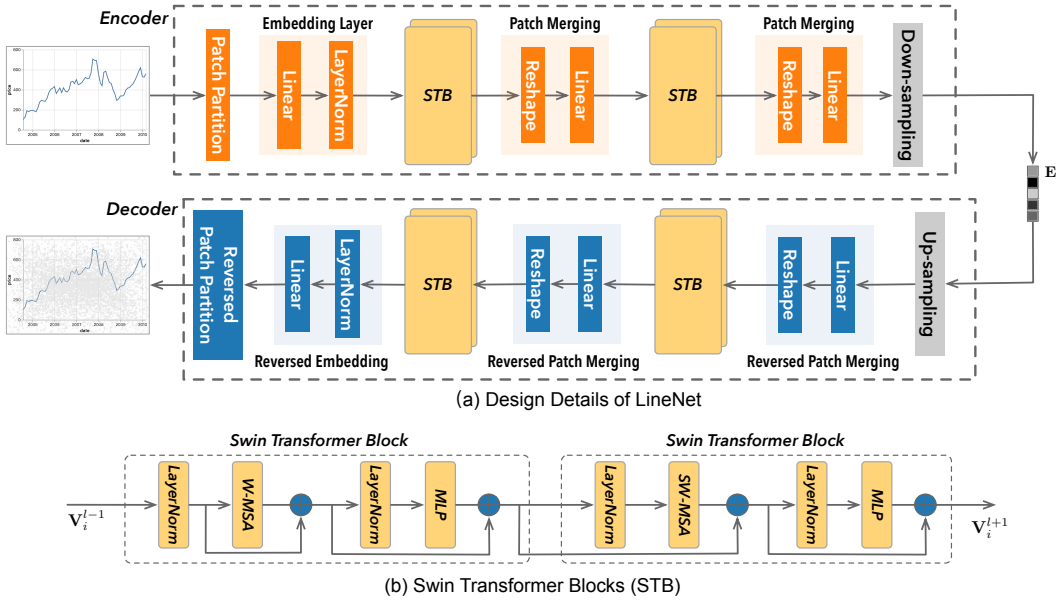


Fig. 4. The architecture of LineNet.

Swin Transformer Blocks. Swin Transformer Blocks (STB) is inspired by the origin Transformer [77], STB introduces a novel local attention and a shifted window mechanism to reduce computation complexity while improving performance.

As shown in Figure 4(b), STB is composed of layer normalization (LayerNorm), multi-head self-attention (MSA), and a 2-layer multi-layer perceptron (MLP). Specifically, STB implements a *window-based multi-head self-attention* (W-MSA) and a *shifted-window-based multi-head self-attention* (SW-MSA) based on the standard multi-head self-attention (MSA) [77]. The former is responsible for performing the self-attention mechanism inside the local window, while the latter shifts the windows to capture additional information between neighbor windows. The overall process of two STB in Figure 4(b) can be summarized as:

$$\overline{\mathbf{V}}_i^l = \text{W} - \text{MSA}(\text{LayerNorm}(\mathbf{V}_i^{l-1})) + \mathbf{V}_i^{l-1}$$
$$\mathbf{V}_i^l = \text{MLP}(\text{LayerNorm}(\overline{\mathbf{V}}_i^l)) + \overline{\mathbf{V}}_i^l$$
$$\overline{\mathbf{V}}_i^{l+1} = \text{SW} - \text{MSA}(\text{LayerNorm}(\mathbf{V}_i^l)) + \mathbf{V}_i^l$$
$$\mathbf{V}_i^{l+1} = \text{MLP}(\text{LayerNorm}(\overline{\mathbf{V}}_i^{l+1})) + \overline{\mathbf{V}}_i^{l+1}$$

Design Details. Figure 4(a) shows the design details of LineNet. In Encoder part, it takes as input a line chart image (*i.e.,* in the form of an image) with size of $3 \times H \times W$. Then, the Patch Partition layer splits the input image into non-overlapping patches with the patch size of 4×8. Next, it treats each image patch as a token and then projects these tokens using a linear embedding layer. That is the embedding layer embeds each patch to a $4 \times 8 \times 4 = 128$-dimensional features, while the Reversed Embedding and Reversed Patch Partition layer in Decoder reverses this process. The projected patched tokens are passed to Swin Transformer blocks to perform the multi-head self attention computation. After the feature representation learning by the Swin Transformer blocks, LineNet performs patches merging to generate hidden feature representation. The feature map resolution increases as the above process repeats. The down-sampling layer is a fully connected

Algorithm 1: Training Algorithm of LineNet

Input: A training set of line chart images $\mathbb{V}$;
Output: The converged LineNet;
1 **for** *number of training iterations* **do**
2 **for** $N \leq$ *number of batches* **do**
3 $\mathbb{B} \leftarrow$ get_mini-batch($\mathbb{V}$);
4 $\mathbb{V}_t \leftarrow$ triplets_selection($\mathbb{B}$);
5 LineNet.forward_propagate($\mathbb{V}_t$);
6 loss $\leftarrow$ LineNet.loss_computation();
7 LineNet.backward_propagate(loss);
8 **return** LineNet;

layer with dimensions $[\frac{H}{16} \times \frac{W}{32} \times 4C, h]$, where h is the dimension of embedding vector. We set the h as 2048. The symmetric Decoder reverses the procedures of the Encoder.

We implement LineNet model with PyTorch and train using AdamW optimizer [49] with the learning rate 5.0×10^{-6}.

Training Algorithm of LineNet. Based on the overall optimization goal, we train LineNet using the process shown in Algorithm 1. It first loops over the number of epochs (Lines 1–7). For each batch (Lines 2-7), it first gets the mini-batch (Line 3) and then selects a set $\mathbb{V}_t$ of triplets from the mini-batch (Line 4), performs forward-propagation (Line 5) to project the embeddings and to reconstruct the inputs, computes the triplet loss of reconstructed using the Eq. (4) (Line 6), and then runs backward-propagation to update the model parameters (Line 7).

4.3 Training Data: Triplets Selection

To learn data-aware image representations of line charts, we need a large number of labeled "similar/dissimilar" line chart images to construct the triplet set for feeding LineNet. Instead of generating triplets through a large number of human annotations, we utilize a set $\mathbb{D}$ of underlying data of a set $\mathbb{V}$ of line chart images, from our constructed LineBench (Section 6), to generate a set of "similar/dissimilar" labels for constructing a set of triplets $\mathbb{V}_t$.

Relevance Measure $R(\mathbf{V}_i, \mathbf{V}_j)$**.** To form a triplet $(\mathbf{V}, \mathbf{V}^+, \mathbf{V}^-)$, we need to select a positive sample $\mathbf{V}^+$ relevant to the anchor $\mathbf{V}$ and select a negative sample $\mathbf{V}^-$ dissimilar to $\mathbf{V}$. Thus, we need to measure relevance between line chart images $\mathbf{V}^+$ and $\mathbf{V}$ (*resp.* $\mathbf{V}^-$ and $\mathbf{V}$). We define the relevance measure between $\mathbf{V}_i$ and $\mathbf{V}_j$ by computing their **dist** on the underlying data $\mathbf{D}_i$ and $\mathbf{D}_j$ of line charts $\mathbf{L}_i$ and $\mathbf{L}_j$: $R(\mathbf{V}_i, \mathbf{V}_j) = 1 - \mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j), (\mathbf{D}_i, \mathbf{D}_j \in \mathbb{D})$.

Therefore, we can utilize the "*data-level similarity*" measure, *i.e.,* $1 - \mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)$, as a proxy for the "*image-level similarity*" of the corresponding line chart images $\mathbf{V}_i$ and $\mathbf{V}_j$.

Based on the above discussion, we now introduce how to select a set $\mathbb{V}_t$ of triplets in a mini-batch during training.

First, for each sample in a mini-batch, we treat it as an anchor $\mathbf{V}_a$. Given an anchor $\mathbf{V}_a$, we first generate a set of candidate positive (*resp.* negative) samples by computing $R(\mathbf{V}_a, \mathbf{V}_i)$ $(\mathbf{V}_i \in \mathbb{V})$ on their underlying data. Intuitively, if a positive sample is already close to the anchor (*i.e.,* their distance on the embedding space is small), then this sample won't help the training much; otherwise, if a positive sample has a large distance from the anchor, it is more "informative" and can better help train the model through back-propagation, and their distance *w.r.t.* their embeddings will be closer after training.

Therefore, we first select a positive sample $\mathbf{V}^+$ if its distance to the anchor is the largest among all candidates. Next, we randomly select a negative sample $\mathbf{V}^-$ if it satisfies with $\textbf{dist}(\mathcal{F}(\mathbf{V}_a), \mathcal{F}(\mathbf{V}^-)) < \beta + \textbf{dist}(\mathcal{F}(\mathbf{V}_a), \mathcal{F}(\mathbf{V}^+))$, where β is a predefined margin. We then construct a triplet $(\mathbf{V}_a, \mathbf{V}^+, \mathbf{V}^-)$ with an anchor, a similar, and a dissimilar line chart image. For example, the triplet $(\mathbf{V}, \mathbf{V}_1^+, \mathbf{V}_1^-)$ in the Figure 3 is a desired triplet. We repeat the above process to form a set $\mathbb{V}_t$ of triplets until all anchors are consumed up. Finally, we use $\mathbb{V}_t$ to train LineNet.

5 DIVERSIFIED TRIPLETS SELECTION

In Section 4.3, we have introduced a training data selection method based on the semi-hard triplets selection strategy [70]. However, this method may not guarantee that the selected triplets are discriminative and meaningful, which may lead to slow convergence and lower performance.

To alleviate this problem, we exploit a diversified triplets selection strategy to carefully pick a small set $\mathbb{V}_t$ of representative and discriminative triplets from current mini-batch $\mathbb{B}$ of training data. Roughly speaking, we hope that the selected triplets can contain more representative line chart images. In other words, these triplets can cover the trends and patterns of different line charts as much as possible. Intuitively, it is beneficial for us to select those line charts with different trends/patterns from each other (*i.e.,* diversified from each other) to construct a set of triplets used to train LineNet, which can reduce the number of triplets, improve training efficiency, and get better performance.

We first introduce some key concepts before we formally introduce the diversified triplets selection algorithm.

Diversity Measure $D(\mathbf{V}_i, \mathbf{V}_j)$. Conceptually, if the distance between line charts in the embedding space is large enough, their visual features and patterns should be greatly different from each one. Hence, we can measure the diversity between line chart images $\mathbf{V}_i$ and $\mathbf{V}_j$ by computing their distance in the embedding space, *i.e.,* $D(\mathbf{V}_i, \mathbf{V}_j) = \textbf{dist}(\mathcal{F}(\mathbf{V}_i), \mathcal{F}(\mathbf{V}_j))$.

If we want to judiciously select a set of diversified triplets, the first step is to pick a set of anchors that are diversified from each other because the selection of positive/negative samples relies on the anchors.

Diversified Anchors Selection. We aim to select k diversified anchors to form $\mathbb{V}_a$. To this end, we first randomly pick a sample from $\mathbb{B}$ as the seed anchor. Next, we select a new anchor $\mathbf{V}_a$ from $\mathbb{B} \setminus \mathbb{V}_a$ if it satisfies:

$$\mathbf{V}_a = \underset{\mathbf{V}_i \in \mathbb{B} \setminus \mathbb{V}_a}{\arg\max} \sum_{\mathbf{V}_j \in \mathbb{V}_a} D(\mathbf{V}_i, \mathbf{V}_j) \tag{5}$$

We repeat the above step until k diversified anchors are derived. The intuition behind this is that we can select a set of anchors far from each other in the embedding space by maximizing the total distance between all anchors in the embedding space.

Alternatively, since computing the diversified anchors using Eq. (5) is time-consuming, we propose to use clustering algorithms [25] (*e.g.,* k-means) to cluster the samples in the mini-batch $\mathbb{B}$ into k clusters $\{C_1, C_2, ..., C_k\}$. Next, we randomly pick one point from each cluster C_i to form the anchor set $\mathbb{V}_a$ for diversity.

Once the k diversified anchors are selected, we utilize the k anchors as seeds to further select a set of representative positive/negative samples to construct triplets. We now formulate the diversified positive/negative samples selection problem as below.

Diversified k Positive/Negative Samples Selection. Given a set of candidate positive (*resp.* negative) samples $\mathbb{V}_p$ (*resp.* $\mathbb{V}_n$) with size n to an anchor $\mathbf{V}_a$, the diversified k positive (*resp.*

negative) samples selection problem aims to pick a set of positive (*resp.* negative) samples $\mathbb{V}'_p \subseteq \mathbb{V}_p$ (*resp.* $\mathbb{V}'_n \subseteq \mathbb{V}_n$) with size k such that:

$$\mathbb{V}'_p = \underset{\widetilde{\mathbb{V}}_p \subseteq \mathbb{V}_p, |\widetilde{\mathbb{V}}_p|=k}{\arg\max} \mathcal{P}(\mathbf{V}_a, \widetilde{\mathbb{V}}_p) \tag{6}$$

$$\mathbb{V}'_n = \underset{\widetilde{\mathbb{V}}_n \subseteq \mathbb{V}_n, |\widetilde{\mathbb{V}}_n|=k}{\arg\max} \mathcal{N}(\mathbf{V}_a, \widetilde{\mathbb{V}}_n) \tag{7}$$

$$\mathcal{P}(\mathbf{V}_a, \widetilde{\mathbb{V}}_p) = \lambda \sum_{\mathbf{V}_i \in \widetilde{\mathbb{V}}_p} [R(\mathbf{V}_i, \mathbf{V}_a) + D(\mathbf{V}_i, \mathbf{V}_a)] + \frac{2(1-\lambda)}{k-1} \sum_{\mathbf{V}_i, \mathbf{V}_j \in \widetilde{\mathbb{V}}_p} D(\mathbf{V}_i, \mathbf{V}_j) \tag{8}$$

$$\mathcal{N}(\mathbf{V}_a, \widetilde{\mathbb{V}}_n) = \lambda \sum_{\mathbf{V}_i \in \widetilde{\mathbb{V}}_n} [(1 - R(\mathbf{V}_i, \mathbf{V}_a)) + (1 - D(\mathbf{V}_i, \mathbf{V}_a))] + \frac{2(1-\lambda)}{k-1} \sum_{\mathbf{V}_i, \mathbf{V}_j \in \widetilde{\mathbb{V}}_n} D(\mathbf{V}_i, \mathbf{V}_j) \tag{9}$$

where $\lambda \in [0, 1)$ is a parameter controlling the *trade-off* between *relevance* and *diversity*, which can be adjusted. The intuition behind this definition is that we aim to maximize $\mathcal{P}(\mathbf{V}_a, \widetilde{\mathbb{V}}_p)$ (*resp.* $\mathcal{N}(\mathbf{V}_a, \widetilde{\mathbb{V}}_n)$) so that we can derive a set of relevant and diversified line chart images with relatively high relevance to the anchor line chart image as well as high diversity. More concretely, the first two terms of $\mathcal{P}(\mathbf{V}_a, \widetilde{\mathbb{V}}_p)$ (*resp.* $\mathcal{N}(\mathbf{V}_a, \widetilde{\mathbb{V}}_n)$) aim to select the semi-hard triplets [70], while the third term adjusts the selection process by considering the *diversity* among positive (*resp.* negative) line charts. We scale down the diversity part by $\frac{2(1-\lambda)}{k-1}$ because there are k items in the first two terms while $\frac{k(k-1)}{2}$ items in the diversity terms.

Unfortunately, the diversified top-k positive/negative samples selection problem is NP-hard. Because our problem can be reduced to the NP-hard *max-sum dispersion* problem [26] when $\lambda = 0$.

Therefore, we devise a greedy algorithm to select top-k positive/negative samples effectively and efficiently, as shown in Algorithm 2 (Lines 9-15). The key idea behind the scenes is that it first greedily constructs a set $\mathbb{V}_a$ of diversified anchors. Next, the algorithm incrementally and greedily selects a positive (*resp.* negative) sample to diversified results $\mathbb{V}'_p$ (*resp.* $\mathbb{V}'_n$).

Diversified Triplets Selection Strategy. Based on the above discussion, we now introduce our diversified triplets selection algorithm. The pseudo code is shown in Algorithm 2. It takes as input a mini-batch of line charts $\mathbb{B}$ (training data), k for the number of anchors (triplets). First, it clusters the $\mathbb{B}$ into k clusters and pick one random point from each cluster to form the diversified anchor set $\mathbb{V}_a$ (Lines 2-5). Next, it iterates each anchor $\mathbf{V}_a$ to select a set of diversified positive/negative samples to anchor $\mathbf{V}_a$ (Lines 6-15). In each loop, it first generates a set of candidate positive/negative samples based on the precomputed pseudo labels, which is straightforward to perform (Line 7). Next, it aims at maximizing the Eq. (10) to select a diversified positive sample and to build the positive samples set $\mathbb{V}'_p$ incrementally (Lines 9-11). This process will be repeated until k positive samples are derived. The selection of diversified negative samples is similar, we omit the discussion due to the space constraint. Finally, it constructs the triplets based on the $\mathbf{V}_a$ and the selected diversified positive (*resp.* negative) set $\mathbb{V}'_p$ (*resp.* $\mathbb{V}'_n$) (Line 15). After iterating all anchors in $\mathbb{V}_a$, the algorithm generates a set $\mathbb{V}_t$ of diversified triplets to drive the training of LineNet.

$$\mathcal{P}'(\mathbf{V}_i, \mathbf{V}_a) = \lambda[R(\mathbf{V}_i, \mathbf{V}_a) + D(\mathbf{V}_i, \mathbf{V}_a)] + \frac{1-\lambda}{k} \sum_{\mathbf{V}_j \in \mathbb{V}'_p} D(\mathbf{V}_i, \mathbf{V}_j) \tag{10}$$

$$\mathcal{N}'(\mathbf{V}_i, \mathbf{V}_a) = \lambda[1 - R(\mathbf{V}_i, \mathbf{V}_a) + 1 - D(\mathbf{V}_i, \mathbf{V}_a)] + \frac{1-\lambda}{k} \sum_{\mathbf{V}_j \in \mathbb{V}'_n} D(\mathbf{V}_i, \mathbf{V}_j) \tag{11}$$

Algorithm 2: DiversifiedTripletsSelection

Input: A mini-batch of line chart $\mathbb{B}$, k;
Output: A mini-batch of diversified triplets $\mathbb{V}_t$;
// 1. Diversified k anchors selection
$\mathbb{V}_t, \mathbb{V}_a \leftarrow \emptyset, \emptyset$; // Initialize the triplet and anchor set
$\{C_1, C_2, ..., C_k\} \leftarrow$ KmeansClustering($\mathbb{B}, k$);
for C_i *in* $\{C_1, C_2, ..., C_k\}$ **do**
  $\mathbf{V}_a \leftarrow$ RandomPick(C_i);
  $\mathbb{V}_a \leftarrow \mathbb{V}_a \cup \{\mathbf{V}_a\}$;
// 2. Diversified top-k triplets selection
for $\boldsymbol{V}_a \in \mathbb{V}_a$ **do**
  // candidate $\mathbb{V}_p$,$\mathbb{V}_n$ generation based on $\mathbf{V}_a$
  $\mathbb{V}_p, \mathbb{V}_n \leftarrow$ CandidateSampleGeneration($\mathbf{V}_a$);
  $\mathbb{V}'_p, \mathbb{V}'_n \leftarrow \emptyset, \emptyset$;
  // diversified positive samples selection
  while $|\mathbb{V}'_p| \leq k$ **do**
    $\mathbf{V}'_p \leftarrow \arg\max_{\mathbf{V}_i \in \mathbb{V}_p} \mathcal{P}'(\mathbf{V}_i, \mathbf{V}_a)$;
    $\mathbb{V}'_p \leftarrow \mathbb{V}'_p \cup \{\mathbf{V}'_p\}, \mathbb{V}_p \leftarrow \mathbb{V}_p \setminus \{\mathbf{V}'_p\}$;
  // diversified negative samples selection
  while $|\mathbb{V}'_n| \leq k$ **do**
    $\mathbf{V}'_n \leftarrow \arg\max_{\mathbf{V}_i \in \mathbb{V}_n} \mathcal{N}'(\mathbf{V}_i, \mathbf{V}_a)$;
    $\mathbb{V}'_n \leftarrow \mathbb{V}'_n \cup \{\mathbf{V}'_n\}, \mathbb{V}_n \leftarrow \mathbb{V}_n \setminus \{\mathbf{V}'_n\}$;
  // Construct diversified triplets
  $\mathbb{V}_t \leftarrow \mathbb{V}_t \cup$ ConstructTriplets($\mathbf{V}_a, \mathbb{V}'_p, \mathbb{V}'_n$);
return $\mathbb{V}_t$;

Overall, our diversified triplets selection strategy will construct a small number of the most discriminative and informative triplets to drive the model training, improve the training efficiency and get better performance, which is verified by the experiments (Section 7).

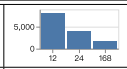
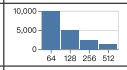
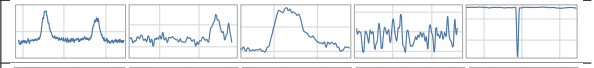
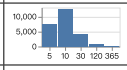
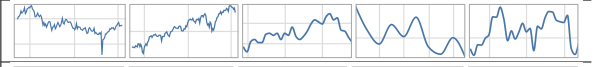
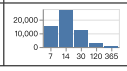
6 LINEBENCH CONSTRUCTION

To facilitate the study of similarity search of line chart visualizations, we need a large-scale line charts corpus to drive the model training and evaluate the search results. Unfortunately, existing visualization corpora [23, 31, 32, 38, 56, 57, 81] can not meet our usage scenario because they are developed for outlier detection, visualization recommendation, etc.

Generally speaking, there are two ways to obtain such a corpus if there is no *off-the-peg* corpus that meets our considerations: (1) The first way is to develop a Web Crawler (*e.g.,* Beagle [8]) to harvest line chart visualizations (*e.g.,* SVG-based, PNG, etc.) from popular visualization communities (*e.g.,* D3, Plotly, etc); and (2) The second way is to collect the source datasets (*e.g.,* time-series datasets) and generate line chart visualizations based on the collected datasets.

We vote for the latter because line charts directly harvested from the Web usually don't have the metadata (**D** and **V**) we need. In this paper, we produce a large-scale line chart visualizations

Table 2. Statistics of LineBench. The column–Dist. of Len. shows the distribution of lengths of line charts.

Datasets	#-Rows	#-Cols	#-Charts	Dist. of Len.	Sample Patterns of Line Charts
TrafficVol	48,205	10	13,798		
EEG	14,981	15	18,551		
Stocks	1,985	85	25,460		
AirQuality	9,358	15	58,127		

corpus, namely LineBench. Each line chart **L** in LineBench has the metadata: the associated source dataset, the underlying data **D** for rendering, and the rendered visualization **V** in the form of an image.

The construction of LineBench consists of three steps: source datasets collection, line chart visualization generation (*i.e.,* generate **D** and **V**), and line chart similarity labeling.

Table 2 overviews the key characteristics of LineBench.

Datasets Collection. We aim to collect the datasets for generating line charts with the following characteristics: (1) the datasets should be produced from real-world scenarios, and (2) the datasets should come from different domains (*e.g.,* stocks market, etc).

Based on the above observations and prior work in similarity search on time-series data [60, 73, 74], we collect four real-world datasets from the well-known UCI repository [6].

(1) TrafficVol [3] collects the hourly traffic volume of Interstate 94 Westbound monitoring by the station 301 from 2012 to 2018.

(2) EEG [2] is a dataset that contains 14 EEG signals collected from a continuous EEG measurement with a duration of 117 seconds, along with an attribute indicating the eye state.

(3) Stocks [5] contains a collection of indicators such as stock prices from the NASDAQ stock market. The dataset spans the time period from 2010 to 2017.

(4) AirQuality [1] has 9,358 records of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device from 2004/03 to 2005/02.

Line Chart Visualizations Generation. Next, we visualize good line charts with two steps based on the above datasets:

(1) Query Generation. Given a dataset, we need to generate a set of good queries **Q** for line charts. The line chart queries should sufficiently describe the intrinsic characteristics (*e.g.,* typical patterns) of the source dataset. Next, we execute each query **Q** over dataset to obtain its underlying data **D**, which is used for rendering the line chart visualization **V**.

(2) Visualization Rendering. We need to visualize the underlying data **D** to obtain its line chart visualization **V**. The visual encodings of visualization **V** should be close to the real user's preferences.

For Query Generation, we use three complementary approaches to generate a large number of line chart queries. First, we utilize the *off-the-shelf* visualization recommendation tools [41, 62] to generate a set of good line chart queries on the dataset. Second, inspired by PEAX [42] and Qetch [60], we also adopt a *segmentation*-based method to split the time-series data into a set of overlapping *windows* with a fixed length l. These *windows* are indeed the line chart queries. The length l of a *window* can be determined by the scenarios of the dataset. For example, for the Stocks dataset, we can set the length as 5 to show its exchange performance of 5 consecutive days. Third, we invite four visualization practitioners to manually explore each dataset and write a set of queries to

visualize the line chart. The four practitioners have expertise in data analysis/visualization and over three years of experience in Python, Tableau, Vega-Lite, Matplotlib, and Seaborn. We combine the queries generated by the above methods and remove the duplicate queries to form our queries set. Next, we execute each query to obtain the underlying data **D** for the line chart, and thus we have a set of underlying data used for rendering the line chart visualizations.

For Visualization Rendering, we use the popular line chart visualization grammars, *i.e.,* Vega-Lite, Matplotlib, and Seaborn, to render the line chart visualization **V** based on the underlying data **D**. We collect a set of popular visual encodings (*e.g.,* color, shape, and size) from the visualization communities (*e.g.,* Vega-Lite Example Gallery) in conjunction with some visual encodings identified by Kim and Heer [36]. Therefore, we have a set of visualization grammars and visual encodings available that can be used to generate line chart visualizations in various styles. Finally, for each line chart underlying data **D**, we randomly combine visualization grammars and visual encodings to render its visualization **V**. As shown in Table 2, we generate 13, 798 line chart visualizations for the TrafficVol dataset. It contains three lengths of line charts to show the trend of the traffic volume in the past 12 hours, 24 hours, and 168 hours (*i.e.,* 7 days). Table 2 shows some patterns of the line chart data for each dataset.

So far, we produce a large-scale line chart visualizations corpus, namely LineBench. As shown in Table 2, LineBench has 13,798, 18,551, 25,460, and 58,127 line chart visualizations (115,936 in total) for TrafficVol, EEG, Stocks, and AirQuality datasets, respectively. For each line chart **L**, it has the metadata: the underlying data for rendering **D**, and the rendered visualization **V**.

Line Chart Visualizations Similarity Labeling. The last step is to create the similarity labels between any pair of line chart visualizations for the evaluation of the search performance. There are 115,936 visualizations in LineBench, the number of pairwise similarity comparisons is $115,936^2$, which is too large to generate pairwise similarity labels for all candidates. Therefore, *we only produce similarity labels for a small set of visualizations, which are treated as the testing queries to evaluate the performance of the similarity search.*

Firstly, we randomly sample 100 line chart visualizations from each dataset as the testing queries. For each line chart visualization $\mathbf{L}_i$, we compute $\mathbf{dist}(\mathbf{D}_i, \mathbf{D}_j)$ using the DTW and ED distance to find a set $\mathbb{L}_i$ of candidates that are similar to the query $\mathbf{L}_i$. More concretely, the candidate set $\mathbb{L}_i$ consists of top-200 similar results measured by DTW and ED distance, respectively, and thus we generate 400 similarity candidates for each query. Finally, we remove the duplicate candidates.

Next, we conduct crowdsourcing study [10, 12] to produce the ground truth from the candidates. We recruit in total 2,869 workers from a crowdsourcing platform MTurk [4]. The workers should have the number of HITs approved $\geq 10,000$ and had $\geq 95\%$ HIT approval rating. We next design a HIT task for the similarity judgments. Each HIT is comprised of ten judgments, and each judgment has a query and a *possibly* similar line chart visualization. We ask the workers to judge whether the two line chart visualizations are similar with five choices: {*very similar, similar, neutral, dissimilar, very dissimilar*}. We assign each HIT to three workers and use the majority voting to aggregate the results. Finally, we aggregate a set of ground truth for each query visualization. We have 100 testing queries (*i.e.,* the query line chart visualizations) for each dataset, and each testing query has averagely ∼132 similar line charts (*i.e.,* ground truth).

7 EXPERIMENT

7.1 Experimental Settings

Methods. We evaluate the following methods:

Table 3. Experimental datasets – LineBench

Datasets	Training		Testing	
	Training	Validation	#-Queries	Repository
TrafficVol	6,899	1,380	100	5,519
EEG	9,274	1,855	100	7,421
Stocks	12,730	2,546	100	10,184
AirQuality	29,063	5,813	100	23,251

(1) LineNet (ours) follows the implementation and triplet selection strategy we introduced in Section 4.
(2) LineNet+ (ours) enhances LineNet by devising a diversified triplet selection strategy we introduced in Section 5.
(3) Qetch [60, 61] is a query-by-sketch system for finding desired patterns of a time series database, by measuring the *data-level similarity*. Qetch designs a matching algorithm by tolerating distortions when performing similarity search on time-series data (https://github.com/dtl-nyuad/qetch).
(4) Zenvisage [73] is a visual query system for searching similar line charts to a given query, by measuring the *data-level similarity*. The similarity algorithm accepts the data points (*i.e.,* the underlying data) of line charts as input. We use the source code (segmentation and MVIP-based matching algorithms) to perform the evaluation (http://zenvisage.github.io/). Note that we only report the best results for the above two methods.
(5) V-CNN [43]. We follow a recent study [43] about visualization retrieval to adapt the ResNet-50 [27] for searching similar line charts. V-CNN considers the *visualization-level* similarity.
(6) Regressor-based Method extracts features of a pair of line chart images by an encoder and maps them into similarity scores. We use the same Swin Transformer Blocks as LineNet to build the model.
(7) Siamese-based Method optimizes the representations between similar line chart images based on a contrastive loss function [35] to reflect their similarity. We also use the same Swin Transformer Blocks as LineNet to implement the model.
(8) Peax [42] is a feature-based system for interactive visual pattern search on sequential data, by measuring the *data-level similarity*. It first extracts features of the data, and based on which involves users to interactively label similarity candidates to train a classifier. Finally, the classifier returns similar results to a query after training.

Note that Zenvisage, Qetch, and Peax take the underlying data **D** of line charts as input for similarity search, while the rest of the methods take the visualization (*i.e.,* in the form of images) **V** of line charts as input.

Environment. All algorithms are implemented by Python 3.6. The neural networks are implemented using Pytorch. All experiments are conducted on a Ubuntu server with 80 cores of Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz and 256GB RAM.

Datasets. We use LineBench to evaluate all methods. For training and testing, we split LineBench into non-overlapping training, validation, and testing set, as shown in Table 3. We use the training set of each dataset to train LineNet, LineNet+, V-CNN, Regressor and Siamese-based methods, respectively. *Note that crowd-sourced similarity labels in* LineBench *can be used to test all methods.*

Evaluation Metrics. We aim to compare the performance of top-k similarity search of line charts. Following previous studies [86, 87] in the top-k similarity search, we use the following metrics: (1) The P@k metric is widely used to measure how many ground truth are captured by top-k search results. We use P@10 to evaluate how many ground truths are captured by the top-10 results.

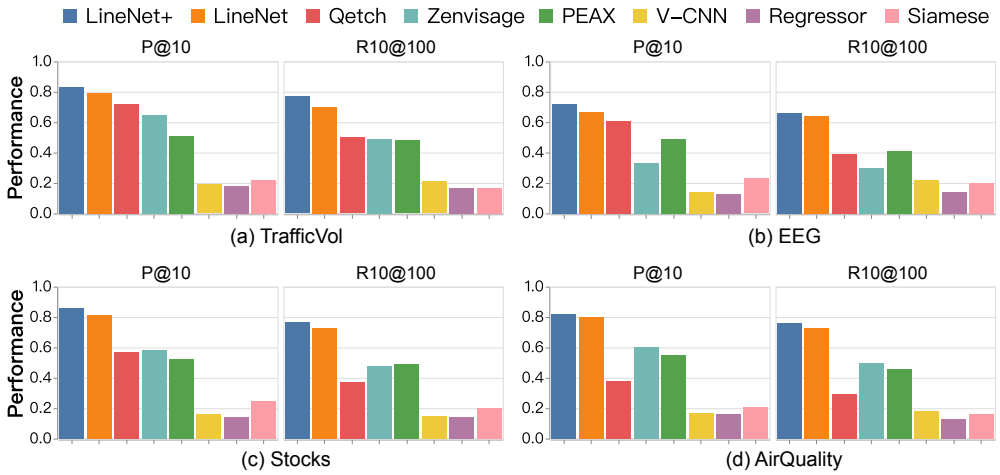


Fig. 5. Comparison with the state of the art.

(2) The Rt@k metric is used to measure the top-t ground truths that are covered by the top-k search results. Specifically, we take R10@100 to show how many top-10 ground truths are captured by the top-100 search results returned by models.

The closer P@10 and R10@100 are to 1, the better the performance of the top-k similarity search.

7.2 Experimental Results

Exp-1: Comparison with the State of the art. We use the 400 testing queries from four datasets in Table 3 to compare LineNet and LineNet+ against state-of-the-art methods. For these 400 testing queries, we use the crowd-sourced ground truth, which is measured through human perception, to evaluate all methods, as introduced in Section 6. We conduct similarity search with each testing query in LineBench's searching repository (Table 3). We follow the user study procedure and instruction of Peax [42] to invite six experts with various backgrounds (*e.g.,* visualization and database) to participate in our user study. Next, we randomly sample 10 testing queries and the corresponding search repository from four datasets, respectively. We then ask participants to interact with Peax to perform the similarity search based on the queries.

Finally, we compute the average P@10 and R10@100 for all testing queries and methods. Figure 5 reports the experimental results. We make the following observations.

(1) LineNet and LineNet+ significantly outperform state-of-the-art methods in four datasets in terms of P@10 and R10@100. The result is expected because LineNet and LineNet+ incorporate the underlying data and rendered images of line charts to holistically and accurately measure the similarity.

(2) LineNet+ achieves better performance than LineNet. For example, LineNet+ achieves an average 0.8075 in terms of the P@10 metric, which outperforms LineNet by 4%. This result shows the effectiveness of the diversified triplets selection strategy.

(3) The third observation is that the methods (*i.e.,* Zenvisage and Qetch) of calculating similarity based on underlying data **D** are better than V-CNN of computing similarity solely by image features. This result validates our discussion in the introduction that relying only on visual features of line charts is difficult to achieve good performance for searching similar line charts. Our methods (*i.e.,* LineNet and LineNet+) consider both the underlying data and visualization features of line charts during training, which can learn a more comprehensive similarity measure for line charts.

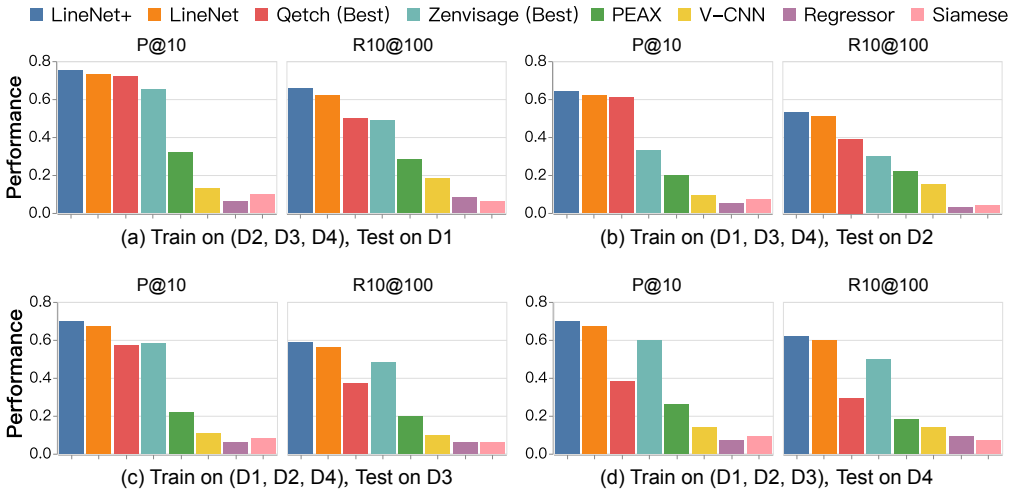


Fig. 6. Overall generalization ability. TrafficVol (D1), EEG (D2), Stocks (D3), AirQuality (D4).

(4) Qetch and Zenvisage achieve better results than PEAX on the TrafficVol and Stocks in terms of P@10, while their performance significantly degrades on the AirQuality. The main reason is that Qetch and Zenvisage may fail to handle the similarity search on some datasets with specific trends and patterns, while the feature-based methods (*i.e.,* LineNet, LineNet+, PEAX) are more stable.
(5) The Regressor-based method does not work well because it is rather hard to accurately compute the similarity score from the large and complicated feature space of line charts based on a regressor. The Siamese-based method is better than Regressor but still fails to learn good representations across similar and dissimilar line charts.
(6) The interactive similarity search method PEAX outperforms V-CNN, Regressor-based and Siamese-based methods. It shows that user feedback is important in the similarity search. In addition, PEAX achieves competitive results compared with Zenvisage and Qetch in term of R10@100. This indicates that incorporating user feedback in searches helps to find ground truths accurately.

Overall, the above results show the effectiveness of our approaches – LineNet and LineNet+.

Exp-2: Evaluation on Generalization Ability. In this set of experiments, we train and test LineNet and LineNet+ across different dataset domains to evaluate the generalization ability and compare with baselines. Specifically, given four dataset domains (TrafficVol, EEG, Stocks, and AirQuality), we evaluate the generalization ability by training LineNet, LineNet+, PEAX, V-CNN, Regressor, and Siamese on three of the four domains and testing on the rest one, respectively. We use the same testing queries as Exp-1 to test their performance on each unseen testing dataset, respectively. Note that since Zenvisage and Qetch adopt data-based similarity algorithms, we report their best results. Figure 6 and Figure 7 show the results.

Figure 6 shows the overall generalization ability of all methods in terms of the average P@10 and R10@100 metrics. For example, Figure 6(a) reports the result of all methods trained on EEG, Stocks, and AirQuality datasets while testing on the TrafficVol dataset.

We make the following observations based on Figure 6.

(1) Compared to the best results in Figure 5, the performance of all learning-based methods degrades to varying degrees. Although the performance of all methods drops in the generalization test, LineNet and LineNet+ still outperform the other methods, especially other

Fig. 7. Detailed generalization (LineNet+).

Table 4. Ablation studies on TrafficVol dataset.

Methods	Overall Results		Generalization	
	P@10	R10@100	P@10	R10@100
LineNet-D	0.74	0.69	0.49	0.38
LineNet-V	0.11	0.09	0.10	0.07
LineNet	**0.79**	**0.70**	**0.73**	**0.63**

learning-based methods, because our solution is equipped with more effective learning framework and training strategy.

(2) LineNet+ outperforms LineNet because LineNet+ is equipped with a sophisticated diversified triplets selection strategy that can help the model better capture the most representative features of the line chart images, which is beneficial to the generalization ability.

In addition, we also study how well our method trained on one dataset can generalize to other datasets. Figure 7 shows the P@10 metric across different training datasets and testing datasets of our method. For example, LineNet+ trained on the training set of TrafficVol can achieve a P@10 score of 0.65 on the testing set of Stocks. We make the following observations based on Figure 7.

(1) Figure 7 shows that the P@10 score of our method indeed drops on the generalization test but still keeps good performance. The generalization ability on EEG dataset is acceptable because EEG is a domain-specific dataset with complex data patterns and trends.
(2) Compared with Figure 6, we can see that our method achieves better results by including more training datasets. It suggests that we can include as many datasets (from different domains) as possible in the training set to enhance the generalization ability.

Exp-3: Ablation Studies on Loss Functions. We conduct this set of ablation studies to evaluate the effects of loss functions of LineNet. Specifically, we investigate two variants of LineNet: (1) LineNet-D: a variant with the triplet loss only. (2) LineNet-V: a variant with the reconstruction loss only. We evaluate the overall performance (similar to Exp-1) and the overall generalization ability (similar to Exp-2) of two variants and compare with LineNet, which is equipped with both reconstruction and triplet loss.

Table 4 reports the results with the following insights.

(1) LineNet-D achieves better performance than LineNet-V since LineNet-D explicitly learns the data similarity between line charts based on the line chart triplets. However, its generalization is poor.
(2) Similar to V-CNN, LineNet-V works poorly on overall test because it is hard to learn useful representations of line charts solely based on the image features. However, the performance of LineNet-V drops less than LineNet-D on the generalization test. The reason is that LineNet-V aims at retaining the most representative visual features of the line charts, which helps to improve the generalization ability across line charts from multiple domains.
(3) LineNet with reconstruction and triplet loss together achieves the best result both on overall and generalization tests. This result is expected because LineNet learns data-aware image representations of line charts based on reconstruction and triplet loss.

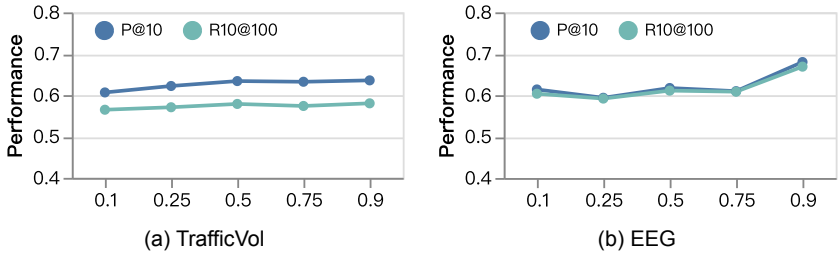


Fig. 8. By varying the parameter α.

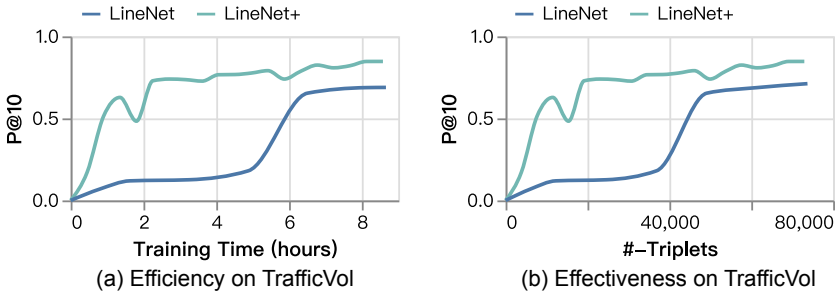


Fig. 9. Efficiency and effectiveness of model training.

Exp-4: Parameters for Trade-off Data and Visual Similarity. This set of experiments aims to analyze the effect of the parameter α for trade-off the data and visual similarity based on Eq. (4). Specifically, we set the range of values of α to $[0.1, 0.25, 0.5, 0.75, 0.9]$. Figure 8 shows the results on TrafficVol and EEG datasets.

(1) We can see that LineNet generally performs better with a larger α. However, it does not achieve the best results if only data-level similarity is considered (*i.e.,* LineNet-D in Table 4).
(2) When varying α, LineNet performs well and is stable. It indicates that LineNet is insensitive to α when α ranges from 0.1 to 0.9.

Exp-5: Efficiency and Effectiveness of Model Training. We study the efficiency and effectiveness of model training.

Efficiency of Training. Figure 9(a) shows LineNet and LineNet+ can converge after about 8 hours, while it takes about 13 and 10 hours to train Regressor-based and Siamese-base methods, respectively. Overall, the training time of our model is acceptable because once the model is well trained, our model can be used *off-the-shelf* for searching similar line charts. Furthermore, the performance of LineNet+ grows very fast at the beginning, reaching competitive results after about 4 hours of training. The reason is that LineNet+ carefully picks a small set of representative triplets based on the diversified triplets selection strategy and will be trained on these representative training data first. Therefore, the diversified triplets selection strategy can reduce the amount of training data, improve training efficiency, and achieve better performance simultaneously. Specifically, the diversified triplets selection algorithm takes about 3.71s per batch to select the triplets, which is only 10.2% of the running time of each batch.

Effectiveness of Training. Figure 9(b) shows that LineNet+ and LineNet converge with about $73k$ triplets, indicating that our model can successfully capture data and visual similarity with a relatively small amount of training triplets. Specifically, we can observe that LineNet+ achieves competitive performance only with $20k$ triplets since it is trained based on a small set of representative triplets.

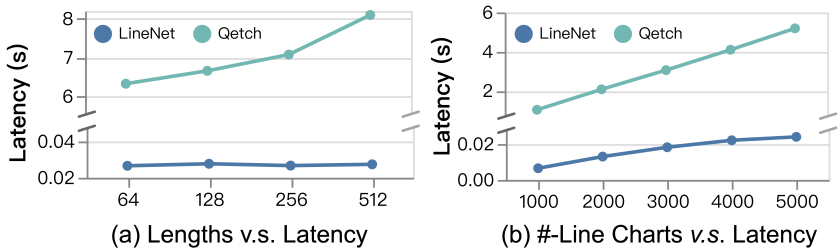


Fig. 10. Search efficiency.

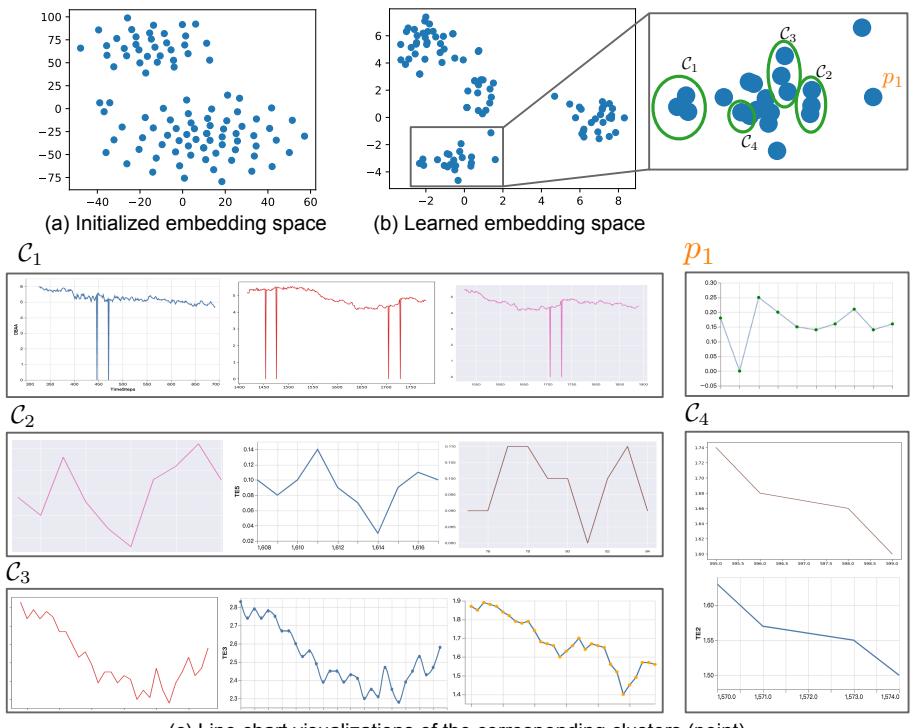


Fig. 11. Visualizations of learned embeddings.

Exp-6: Search Efficiency. We now study the search efficiency of our methods and the competitive baseline–Qetch. Figure 10 shows the experimental results.

Impact of lengths of line charts. As shown in Figure 10(a), we vary the lengths of line charts, *i.e.,* the number of data points in a line chart, to compute the latency of getting top-100 results on the EEG dataset. With the increase in lengths, the overall runtimes significantly increase for Qetch because of the increase in the number of comparisons in a line chart's segmentations by Qetch. In contrast, LineNet is not sensitive to this factor because the efficiency of top-k similarity search is limited by the dimension of the embedding vector and the cost of nearest neighbor search.

Impact of the number of line charts. We study the search scalability of LineNet by varying the number of line charts to be retrieved. Figure 10(b) shows the evaluation results of finding top-100 results in the EEG dataset. Although the overall runtime for LineNet and Qetch grows linearly with the number of line charts, the gap between LineNet and Qetch is large. Overall, LineNet is efficient enough to support the similarity search of line charts.

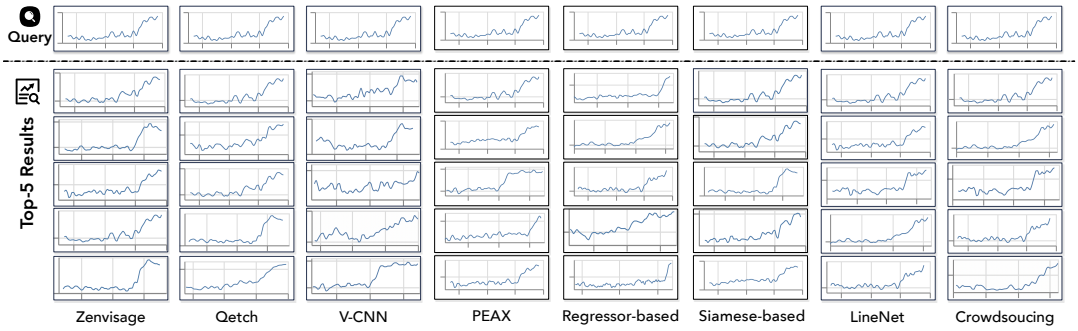


Fig. 12. Case study on LineBench (EEG dataset).

Exp-7: Visualizations of the Learned Embeddings Space. To verify whether the learned representation is discriminative, we use the t-SNE [76] tool to visualize the distribution of line chart visualizations' embeddings generated by LineNet, on the Stocks dataset. Since directly plotting the entire dataset is too cluttered, we use 100 random samples for plotting. We compute the embeddings of these 100 line chart visualizations by the initialized LineNet (*i.e.,* before training) and the trained LineNet (*i.e.,* after training), respectively.

Figure 11(a) shows the distribution of line chart visualizations in their initialized embedding space. We can observe that most of the points are scattered. Figure 11(b) depicts the distribution of line chart visualization embeddings learned by LineNet. We also zoom in on the space and show the corresponding line charts in Figure 11(c). *It clearly shows that visually similar line chart visualizations are closer together in the embedding space, while dissimilar line charts are farther from each other in the embedding space.*

Exp-8: Case Study. In this group of experiments, we show some testing cases used in the previous quantitative evaluation and conduct an additional case study using real-world cases.

Case study on LineBench. Figure 12 shows the top-5 results to the query line chart of different methods on the EEG dataset. Note that we redraw the line chart using Vega-Lite for a better representation. Overall, all methods are able to find similar results to the query but may fall short of capturing some discriminative features of line charts in some cases.

We make the following observations based on Figure 12.

(1) Zenvisage and Qetch can retrieve a set of similar line charts based on the distance metric on the underlying data. However, not all results are visually similar to the line-chart query.
(2) V-CNN solely relies on the visual features of the image to measure the image-level similarity and fails to capture the fine-grained similarity between line charts.
(3) Compared to Zenvisageand Qetch, PEAX outperforms them in measuring the similarity of line charts based on underlying data, thanks in part to PEAX's innovative approach of incorporating user feedback during the search process.
(4) Our method outperforms competitive baselines by incorporating underlying data and visualization characteristics during the training step. This idea enables our method to successfully identify a list of candidate line charts that exhibit fine-grained similarity to the query.

Case study on real-world application scenarios. We apply LineNet, trained by TrafficVol dataset, to search similar line chart images to the query image in the wild. We plan to compare the performance of LineNet against Google Image Search and Bing Image Search to gain more insights. We harvest 2 line chart images from Plotly Gallery (https://plotly.com/python/time-series/) as the query images. Next, we upload each query line chart image to Google Image Search and Bing Image Search and

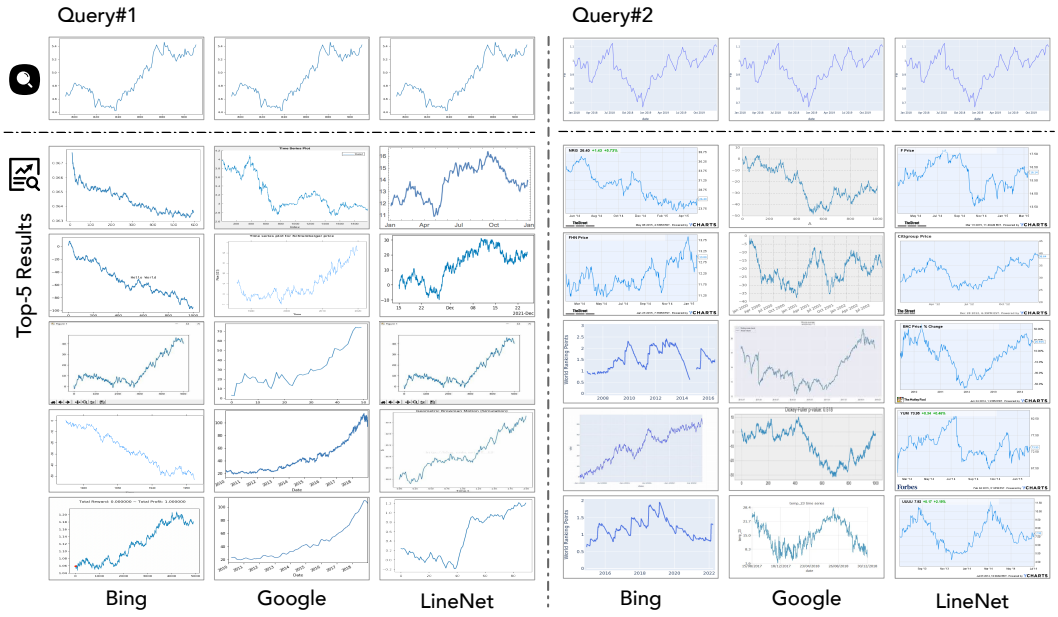


Fig. 13. Case study on real cases.

harvest the top-50 similar images, respectively. Then, we take these images as the search repository and use the same query to perform the similarity search using LineNet.

Figure 13 shows the top-5 search results to the query. We make the following observations.

(1) All methods can retrieve a subset of similar line chart images to the query. However, Google and Bing Image Search return some "bad" cases among the top-5 results. For example, top-2 results from Bing are very dissimilar to the Query#1. The results are reasonable since Bing and Google are not optimized for searching similar line chart images. LineNet works better in this task because our method captures fine-grained features of the query line chart.
(2) LineNet can retrieve similar line chart images in the wild, which is a complementary approach to the existing image search system.

8 RELATED WORK

8.1 Similarity Measures of Line Charts

Similarity measures for measuring the similarity between line charts have been extensively studied [9, 17, 18, 21, 33, 44, 73, 74, 80]. These measures can be broadly divided into two categories: data-based and feature-based similarity measures.

Data-based similarity measures. This line of research aims to quantify the "similarity" by computing the distance between the underlying data **D** of line charts. Dynamic Time Warping (DTW) [17] and Euclidean Distance (ED) [21] are the two most widely used measures. Ding et al. [17] empirically found DTW is one of the systematically *best* distance measures, which is also observed from data mining and visualization communities [37, 60, 67, 85, 88]. In addition, there are also some tail-made distance-based algorithms that are developed to fix distortion errors [60] or improve search efficiency [73, 74].

Feature-based similarity measures. Piecewise Aggregate Approximation (PAA) [33] and Symbolic Aggregate approXimation (SAX) [44] are popular and effective methods that can discretize

time-series data into a set of symbolic features used to measure the similarity. Recently, representation learning [9, 13, 14, 46, 47, 65, 84] has been demonstrated its superior ability in learning discriminative features of time-series data [42, 80]. LineNet falls into this category.

8.2 Similarity Search of Line Charts

There are two lines of work to conduct similar line charts search either based on the underlying data **D** (*data-level* similarity) [28, 40, 42, 60, 61, 68, 73, 74] or according to the rendered line chart visualization **V** (*visualization-level* similarity a.k.a. *image-level* similarity) [30, 43, 64, 69].

Data-level similarity search of line charts utilizes **D** to measure data-level similarity between line charts based on data-based or feature-based similarity measures [28, 40, 42, 60, 61, 68, 73, 74, 80].

Data-based similarity measures. Generally speaking, these tools [28, 60, 61, 68, 73, 74] take as input the data $\mathbf{D}_q$ of the query line chart and the data $\mathbb{D} = \{\mathbf{D}_1, \mathbf{D}_2, ..., \mathbf{D}_n\}$ of candidate line charts. Next, they equip tailor-made similarity algorithms in conjunction with existing distance functions to compute the similarity between $\mathbf{D}_q$ and $\mathbf{D}_i \in \mathbb{D}$. Finally, these tools return a list of most similar line charts to the query $\mathbf{D}_q$, and then render as visualizations to users.

Feature-based similarity measures. The second category [42, 80] leverages deep learning techniques to capture more reliable features from the **D** to facilitate the similarity search. For example, Peax [42] employs a convolutional autoencoder to learn the representations from the **D** and then trains a classifier to interactively find the similar line charts to the $\mathbf{D}_q$. Our proposal utilizes deep representation learning to extract discriminative features for measuring the similarity between line charts. One of the key benefits of our approach is that it only relies on the rendered images **V** and does not require the underlying data **D** at query time. This important feature significantly extends the scope of use scenarios and enhances the practicality of our approach.

Visualization-level similarity search of line charts measure the visualization-level similarity between line charts by leveraging the techniques of representation learning [30, 43, 64, 69]. For example, Li et al. [43] propose to retrieve perceptually similar visualizations by giving priority to the structure information (*e.g.,* the color, the opacity, and the stroke width of the element) of visualizations derived from the Scalable Vector Graphic (SVG), while our method pays more attention to the underlying data of the line chart visualizations (*i.e.,* data-aware visualization similarity). Moreover, Li et al. [43] only support visualizations in the format of Scalable Vector Graphic (SVG), but fall short of supporting the most common formats type–raster graphics [8, 43], while our method can naturally support both vector and raster graphics.

9 CONCLUSION

We present LineNet, a novel Vision Transformer-based Triplet Autoencoder framework that can learn useful representations of line charts to holistically measure the data-level and image-level similarity of line charts. We further propose a diversified triplets selection strategy to optimize the training process and achieve better performance. We also construct a large-scale line chart corpus LineBench to support the task of similarity search. Extensive quantitative evaluations and case studies verify the effectiveness of LineNet.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments. This work is supported by NSF of China (61925205, 62232009, 62102215), Zhejiang Lab's International Talent Fund for Young Professionals, Huawei, BNRist, and TAL Education.

REFERENCES

[1] March 27, 2023. Air Quality Data Set. https://archive.ics.uci.edu/ml/datasets/air+quality

[2] March 27, 2023. EEG Eye State Data Set. https://archive.ics.uci.edu/ml/datasets/EEG+Eye+State

[3] March 27, 2023. Metro Interstate Traffic Volume Data Set. https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume

[4] March 27, 2023. Mturk. https://www.mturk.com/

[5] March 27, 2023. Stocks NASDAQ. https://archive.ics.uci.edu/ml/datasets/CNNpred%3A+CNN-based+stock+market+prediction+using+a+diverse+set+of+variables

[6] March 27, 2023. UCI repository. https://archive.ics.uci.edu/ml/datasets/

[7] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. 1998. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)* 45, 6 (1998), 891–923.

[8] Leilani Battle, Peitong Duan, Zachery Miranda, Dana Mukusheva, Remco Chang, and Michael Stonebraker. 2018. Beagle: Automated Extraction and Interpretation of Visualizations from the Web. In *CHI*, Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox (Eds.). ACM, 594. https://doi.org/10.1145/3173574.3174168

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[10] Chengliang Chai, Lei Cao, Guoliang Li, Jian Li, Yuyu Luo, and Samuel Madden. 2020. Human-in-the-loop Outlier Detection. In *SIGMOD*. ACM, 19–33. https://doi.org/10.1145/3318464.3389772

[11] Chengliang Chai, Guoliang Li, Ju Fan, and Yuyu Luo. 2020. Crowdsourcing-based Data Extraction from Visualization Charts. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 1814–1817.

[12] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2016. Cost-Effective Crowdsourced Entity Resolution: A Partial-Order Approach. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 969–984. https://doi.org/10.1145/2882903.2915252

[13] Chengliang Chai, Jiabin Liu, Nan Tang, Guoliang Li, and Yuyu Luo. 2022. Selective Data Acquisition in the Wild for Model Charging. *Proc. VLDB Endow.* 15, 7 (jun 2022), 1466–1478. https://doi.org/10.14778/3523210.3523223

[14] Chengliang Chai, Jiayi Wang, Yuyu Luo, Zeping Niu, and Guoliang Li. 2022. Data Management for Machine Learning: A Survey. *IEEE Transactions on Knowledge and Data Engineering* (2022), 1–1. https://doi.org/10.1109/TKDE.2022.3148237

[15] Sarah Cohen, Chengkai Li, Jun Yang, and Cong Yu. 2011. Computational Journalism: A Call to Arms to Database Researchers. In *Fifth Biennial Conference on Innovative Data Systems Research, CIDR 2011, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*. www.cidrdb.org, 148–151. http://cidrdb.org/cidr2011/Papers/CIDR11_Paper17.pdf

[16] Kenny Davila, Srirangaraj Setlur, David S. Doermann, Bhargava Urala Kota, and Venu Govindaraju. 2021. Chart Mining: A Survey of Methods for Automated Chart Analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* 43, 11 (2021), 3799–3819. https://doi.org/10.1109/TPAMI.2020.2992028

[17] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.* 1, 2 (2008), 1542–1552. https://doi.org/10.14778/1454159.1454226

[18] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.* 1, 2 (2008), 1542–1552. https://doi.org/10.14778/1454159.1454226

[19] Ensheng Dong, Hongru Du, and Lauren Gardner. 2020. An interactive web-based dashboard to track COVID-19 in real time. *The Lancet infectious diseases* 20, 5 (2020), 533–534.

[20] 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).

[21] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1994. Fast Subsequence Matching in Time-Series Databases. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA, May 24-27, 1994*. ACM Press, 419–429. https://doi.org/10.1145/191839.191925

[22] Arlen Fan, Yuxin Ma, Michelle Mancenido, and Ross Maciejewski. 2022. Annotating Line Charts for Addressing Deception. In *CHI '22: CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, 29 April 2022 - 5 May 2022*, Simone D. J. Barbosa, Cliff Lampe, Caroline Appert, David A. Shamma, Steven Mark Drucker, Julie R. Williamson, and Koji Yatani (Eds.). ACM, 80:1–80:12. https://doi.org/10.1145/3491102.3502138

[23] Yi Guo, Shunan Guo, Zhuochen Jin, Smiti Kaul, David Gotz, and Nan Cao. 2020. Survey on Visual Analysis of Event Sequence Data. *CoRR* abs/2006.14291 (2020). arXiv:2006.14291 https://arxiv.org/abs/2006.14291

[24] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. 2020. A survey on visual transformer. *arXiv e-prints* (2020), arXiv–2012.

[25] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)* 28, 1 (1979), 100–108.

[26] Refael Hassin, Shlomi Rubinstein, and Arie Tamir. 1997. Approximation algorithms for maximum dispersion. *Operations research letters* 21, 3 (1997), 133–137.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*. Springer, 630–645.

[28] Harry Hochheiser and Ben Shneiderman. 2002. Visual queries for finding patterns in time series data. *University of Maryland, Computer Science Dept. Tech Report, CS-TR* 4365 (2002), 18.

[29] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using Triplet network. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6622

[30] Enamul Hoque and Maneesh Agrawala. 2020. Searching the Visual Style and Structure of D3 Visualizations. *IEEE Trans. Vis. Comput. Graph.* 26, 1 (2020), 1236–1245. https://doi.org/10.1109/TVCG.2019.2934431

[31] Kevin Zeng Hu, Snehalkumar (Neil) S. Gaikwad, Madelon Hulsebos, Michiel A. Bakker, Emanuel Zgraggen, César A. Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çagatay Demiralp. 2019. VizNet: Towards A Large-Scale Visualization Learning and Benchmarking Repository. In *CHI*, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 662. https://doi.org/10.1145/3290605.3300892

[32] Petra Isenberg, Florian Heimerl, Steffen Koch, Tobias Isenberg, Panpan Xu, Charles D. Stolper, Michael Sedlmair, Jian Chen, Torsten Möller, and John T. Stasko. 2017. Vispubdata.org: A Metadata Collection About IEEE Visualization (VIS) Publications. *IEEE Trans. Vis. Comput. Graph.* 23, 9 (2017), 2199–2206. https://doi.org/10.1109/TVCG.2016.2615308

[33] Eamonn J. Keogh, Kaushik Chakrabarti, Michael J. Pazzani, and Sharad Mehrotra. 2001. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowl. Inf. Syst.* 3, 3 (2001), 263–286. https://doi.org/10.1007/PL00011669

[34] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. 2021. Transformers in vision: A survey. *ACM Computing Surveys (CSUR)* (2021).

[35] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. *Advances in Neural Information Processing Systems* 33 (2020), 18661–18673.

[36] Younghoon Kim and Jeffrey Heer. 2018. Assessing Effects of Task and Data Distribution on the Effectiveness of Visual Encodings. *Comput. Graph. Forum* 37, 3 (2018), 157–167. https://doi.org/10.1111/cgf.13409

[37] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2021. A Survey on Advancing the DBMS Query Optimizer: Cardinality Estimation, Cost Model, and Plan Enumeration. *Data Sci. Eng.* 6, 1 (2021), 86–101. https://doi.org/10.1007/s41019-020-00149-7

[38] Shahid Latif and Fabian Beck. 2019. VIS Author Profiles: Interactive Descriptions of Publication Records Combining Text and Visualization. *IEEE Trans. Vis. Comput. Graph.* 25, 1 (2019), 152–161. https://doi.org/10.1109/TVCG.2018.2865022

[39] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. 2010. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE, 253–256.

[40] Doris Jung Lin Lee, John Lee, Tarique Siddiqui, Jaewoo Kim, Karrie Karahalios, and Aditya G. Parameswaran. 2020. You can't always sketch what you want: Understanding Sensemaking in Visual Query Systems. *IEEE Trans. Vis. Comput. Graph.* 26, 1 (2020), 1267–1277. https://doi.org/10.1109/TVCG.2019.2934666

[41] Doris Jung Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A. Hearst, and Aditya G. Parameswaran. 2021. Lux: Always-on Visualization Recommendations for Exploratory Dataframe Workflows. *Proc. VLDB Endow.* 15, 3 (2021), 727–738. http://www.vldb.org/pvldb/vol15/p727-lee.pdf

[42] 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. *Comput. Graph. Forum* 39, 3 (2020), 167–179. https://doi.org/10.1111/cgf.13971

[43] 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* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 409, 14 pages. https://doi.org/10.1145/3491102.3502048

[44] Jessica Lin, Eamonn J. Keogh, Stefano Lonardi, and Bill Yuan-chi Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD 2003, San Diego, California, USA, June 13, 2003*. ACM, 2–11. https://doi.org/10.1145/882082.882086

[45] Kaiyi Lin, Xing Xu, Lianli Gao, Zheng Wang, and Heng Tao Shen. 2020. Learning Cross-Aligned Latent Embeddings for Zero-Shot Cross-Modal Retrieval. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*. AAAI

Press, 11515–11522. https://ojs.aaai.org/index.php/AAAI/article/view/6817

[46] Jiabin Liu, Chengliang Chai, Yuyu Luo, Yin Lou, Jianhua Feng, and Nan Tang. 2022. Feature Augmentation with Reinforcement Learning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 3360–3372. https://doi.org/10.1109/ICDE53745.2022.00317

[47] Jiabin Liu, Fu Zhu, Chengliang Chai, Yuyu Luo, and Nan Tang. 2021. Automatic Data Acquisition for Deep Learning. *Proc. VLDB Endow.* 14, 12 (2021), 2739–2742. https://doi.org/10.14778/3476311.3476333

[48] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10012–10022.

[49] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=Bkg6RiCqY7

[50] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. 2020. Interactive Cleaning for Progressive Visualization through Composite Questions. In *36th IEEE International Conference on Data Engineering, ICDE*. IEEE, 733–744.

[51] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. 2020. VisClean: Interactive Cleaning for Progressive Visualization. *Proc. VLDB Endow.* 13, 12 (2020), 2821–2824.

[52] Yuyu Luo, Wenbo Li, Tianyu Zhao, Xiang Yu, Lixi Zhang, Guoliang Li, and Nan Tang. 2020. DeepTrack: Monitoring and Exploring Spatio-Temporal Data - A Case of Tracking COVID-19 -. *Proc. VLDB Endow.* 13, 12 (2020), 2841–2844. https://doi.org/10.14778/3415478.3415489

[53] Yuyu Luo, Xuedi Qin, Chengliang Chai, Nan Tang, Guoliang Li, and Wenbo Li. 2022. Steerable Self-Driving Data Visualization. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (2022), 475–490. https://doi.org/10.1109/TKDE.2020.2981464

[54] Yuyu Luo, Xuedi Qin, Nan Tang, and Guoliang Li. 2018. DeepEye: Towards Automatic Data Visualization. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. 101–112.

[55] Yuyu Luo, Xuedi Qin, Nan Tang, Guoliang Li, and Xinran Wang. 2018. DeepEye: Creating Good Data Visualizations by Keyword Search. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 1733–1736. https://doi.org/10.1145/3183713.3193545

[56] Yuyu Luo, Jiawei Tang, and Guoliang Li. 2021. nvBench: A Large-Scale Synthesized Dataset for Cross-Domain Natural Language to Visualization Task. *CoRR* abs/2112.12926 (2021). arXiv:2112.12926 https://arxiv.org/abs/2112.12926

[57] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing Natural Language to Visualization (NL2VIS) Benchmarks from NL2SQL Benchmarks. In *SIGMOD '21: International Conference on Management of Data, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 1235–1247. https://doi.org/10.1145/3448016.3457261

[58] Yuyu Luo, Nan Tang, Guoliang Li, Wenbo Li, Tianyu Zhao, and Xiang Yu. 2020. DeepEye: A Data Science System for Monitoring and Exploring COVID-19 Data. *IEEE Data Eng. Bull.* 43, 2 (2020), 121–132. http://sites.computer.org/debull/A20june/p121.pdf

[59] Yuyu Luo, Nan Tang, Guoliang Li, Jiawei Tang, Chengliang Chai, and Xuedi Qin. 2022. Natural Language to Visualization by Neural Machine Translation. *IEEE Trans. Vis. Comput. Graph.* 28, 1 (2022), 217–226. https://doi.org/10.1109/TVCG.2021.3114848

[60] 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, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox (Eds.). ACM, 388. https://doi.org/10.1145/3173574.3173962

[61] Miro Mannino and Azza Abouzied. 2018. Qetch: Time Series Querying with Expressive Sketches. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 1741–1744. https://doi.org/10.1145/3183713.3193547

[62] Dominik Moritz, Chenglong Wang, Greg L. Nelson, Halden Lin, Adam M. Smith, Bill Howe, and Jeffrey Heer. 2019. Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco. *IEEE Trans. Vis. Comput. Graph.* 25, 1 (2019), 438–448. https://doi.org/10.1109/TVCG.2018.2865240

[63] Michael Oppermann, Robert Kincaid, and Tamara Munzner. 2021. VizCommender: Computing Text-Based Similarity in Visualization Repositories for Content-Based Recommendations. *IEEE Trans. Vis. Comput. Graph.* 27, 2 (2021), 495–505. https://doi.org/10.1109/TVCG.2020.3030387

[64] Chunyao Qian, Shizhao Sun, Weiwei Cui, Jian-Guang Lou, Haidong Zhang, and Dongmei Zhang. 2021. Retrieve-Then-Adapt: Example-based Automatic Generation for Proportion-related Infographics. *IEEE Trans. Vis. Comput. Graph.* 27,

2 (2021), 443–452. https://doi.org/10.1109/TVCG.2020.3030448

[65] Xuedi Qin, Chengliang Chai, Nan Tang, Jian Li, Yuyu Luo, Guoliang Li, and Yaoyu Zhu. 2022. Synthesizing Privacy Preserving Entity Resolution Datasets. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2359–2371. https://doi.org/10.1109/ICDE53745.2022.00222

[66] Xuedi Qin, Yuyu Luo, Nan Tang, and Guoliang Li. 2020. Making data visualization more efficient and effective: a survey. *VLDB J.* 29, 1 (2020), 93–117.

[67] Thanawin Rakthanmanon, Bilson J. L. Campana, Abdullah Mueen, Gustavo E. A. P. A. Batista, M. Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn J. Keogh. 2012. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*. ACM, 262–270. https://doi.org/10.1145/2339530.2339576

[68] Kathy Ryall, Neal Lesh, Tom Lanning, Darren Leigh, Hiroaki Miyashita, and Shigeru Makino. 2005. QueryLines: approximate query for visual browsing. In *Extended Abstracts Proceedings of the 2005 Conference on Human Factors in Computing Systems, CHI 2005, Portland, Oregon, USA, April 2-7, 2005*, Gerrit C. van der Veer and Carolyn Gale (Eds.). ACM, 1765–1768. https://doi.org/10.1145/1056808.1057017

[69] Babak Saleh, Mira Dontcheva, Aaron Hertzmann, and Zhicheng Liu. 2015. Learning style similarity for searching infographics. In *Proceedings of the 41st Graphics Interface Conference, Halifax, NS, Canada, June 3-5, 2015*, Hao (Richard) Zhang and Tony Tang (Eds.). ACM, 59–64. http://dl.acm.org/citation.cfm?id=2788902

[70] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 815–823. https://doi.org/10.1109/CVPR.2015.7298682

[71] Leixian Shen, Enya Shen, Yuyu Luo, Xiaocong Yang, Xuming Hu, Xiongshuai Zhang, Zhiwei Tai, and Jianmin Wang. 2021. Towards Natural Language Interfaces for Data Visualization: A Survey. *CoRR* abs/2109.03506 (2021). arXiv:2109.03506 https://arxiv.org/abs/2109.03506

[72] Leixian Shen, Enya Shen, Zhiwei Tai, Yun Wang, Yuyu Luo, and Jianmin Wang. 2022. GALVIS: Visualization Construction through Example-Powered Declarative Programming. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, Mohammad Al Hasan and Li Xiong (Eds.). ACM, 4975–4979. https://doi.org/10.1145/3511808.3557159

[73] Tarique Siddiqui, John Lee, Albert Kim, Edward Xue, Xiaofo Yu, Sean Zou, Lijin Guo, Changfeng Liu, Chaoran Wang, Karrie Karahalios, and Aditya G. Parameswaran. 2017. Fast-Forwarding to Desired Visualizations with Zenvisage. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2017/papers/p43-siddiqui-cidr17.pdf

[74] Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, and Aditya G. Parameswaran. 2020. ShapeSearch: A Flexible and Efficient System for Shape-based Exploration of Trendlines. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 51–65. https://doi.org/10.1145/3318464.3389722

[75] Jiawei Tang, Yuyu Luo, Mourad Ouzzani, Guoliang Li, and Hongyang Chen. 2022. Sevi: Speech-to-Visualization through Neural Machine Translation. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 2353–2356. https://doi.org/10.1145/3514221.3520150

[76] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[77] 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).

[78] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. 2014. Learning Fine-Grained Image Similarity with Deep Ranking. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 1386–1393. https://doi.org/10.1109/CVPR.2014.180

[79] Qianwen Wang, Zhutian Chen, Yong Wang, and Huamin Qu. 2021. A Survey on ML4VIS: Applying MachineLearning Advances to Data Visualization. *IEEE Transactions on Visualization and Computer Graphics* (2021), 1–1. https://doi.org/10.1109/TVCG.2021.3106142

[80] Qitong Wang and Themis Palpanas. 2021. Deep Learning Embeddings for Data Series Similarity Search. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, Feida Zhu, Beng Chin Ooi, and Chunyan Miao (Eds.). ACM, 1708–1716. https://doi.org/10.1145/3447548.3467317

[81] Yang Wang, Minzhu Yu, Guihua Shan, Han-Wei Shen, and Zhonghua Lu. 2019. VISPubComPAS: a comparative analytical system for visualization publication data. *J. Vis.* 22, 5 (2019), 941–953. https://doi.org/10.1007/s12650-019-00585-2

[82] Aoyu Wu, Wai Tong, Haotian Li, Dominik Moritz, Yong Wang, and Huamin Qu. 2022. ComputableViz: Mathematical Operators as a Formalism for Visualisation Processing and Analysis. In *CHI Conference on Human Factors in Computing Systems*. 1–15.

[83] Aoyu Wu, Yun Wang, Xinhuan Shu, Dominik Moritz, Weiwei Cui, Haidong Zhang, Dongmei Zhang, and Huamin Qu. 2021. AI4VIS: Survey on Artificial Intelligence Approaches for Data Visualization. *IEEE Transactions on Visualization and Computer Graphics* (2021), 1–1. https://doi.org/10.1109/TVCG.2021.3099002

[84] Ruofan Wu, Feng Zhang, Jiawei Guan, Zhen Zheng, Xiaoyong Du, and Xipeng Shen. 2022. DREW: Efficient Winograd CNN Inference with Deep Reuse. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. ACM, New York, NY, USA, 1807–1816. https://doi.org/10.1145/3485447.3511985

[85] Sai Wu, Ying Li, Haoqi Zhu, Junbo Zhao, and Gang Chen. 2022. Dynamic Index Construction with Deep Reinforcement Learning. *Data Sci. Eng.* 7, 2 (2022), 87–101. https://doi.org/10.1007/s41019-022-00186-4

[86] Xing Xu, Kaiyi Lin, Huimin Lu, Lianli Gao, and Heng Tao Shen. 2020. Correlated features synthesis and alignment for zero-shot cross-modal retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1419–1428.

[87] Peilun Yang, Hanchen Wang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2021. T3S: Effective Representation Learning for Trajectory Similarity Computation. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2183–2188. https://doi.org/10.1109/ICDE51399.2021.00221

[88] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. 2022. Cost-based or Learning-based? A Hybrid Query Optimizer for Query Plan Selection. *Proc. VLDB Endow.* 15, 13 (2022), 3924–3936. https://www.vldb.org/pvldb/vol15/p3924-li.pdf

Received July 2022; revised October 2022; accepted November 2022