J Vis
https://doi.org/10.1007/s12650-022-00887-y

REGULAR PAPER

**Jie Li · Yongjian Sun · Zhenhuan Lei · Siming Chen · Gennady Andrienko · Natalia Andrienko · Wei Chen**

# A hybrid prediction and search approach for flexible and efficient exploration of big data

Received: 2 August 2022 / Accepted: 22 September 2022


**Abstract** This paper presents a hybrid prediction and search approach (HPS) for building visualization systems of big data. The basic idea is training a regression model to predict a coarse range on the dataset and then searching target records that satisfy the query conditions within the range. The prediction reduces the storage cost without preprocessing a data structure storing aggregate values of queriable attribute range combinations. Meanwhile, the search eliminates the prediction bias inevitable for machine learning models. Experiments on multiple open datasets demonstrate HPS's comparable query speed to existing techniques and the potential of continuous performance improvement by investing more hardware resources. In addition, the feature of returning original records instead of aggregate values brings better use flexibility, enabling to construct visualization systems with display/query functions that are unavailable for existing techniques.
**Keywords** Interactive data exploration · Visual query · Machine learning · Neural network · Visual analytics

## 1 Introduction

Interactive data exploration (IDE) is a traditional research topic in visualization. In a typical scenario, the user selects value ranges on attributes to generate views showing aggregate patterns of filtered records, as in Fig. 1a. When the dataset is large, the query latency becomes long due to time-consuming record traversals.

There are many techniques for accelerating the workflow by preprocessing a data structure to store aggregate values of queriable attribute range combinations (Liu et al. 2013; Lins et al. 2013; Pahins et al. 2016; Mei et al. 2019; Moritz et al. 2019). Correct values can be fetched from the data structure immediately, thus achieving efficient IDE by avoiding any record traversal, as in Fig. 1b. However, the prestoring strategy suffers from the problem of high storage overhead. On the other hand, the success of deep learning has inspired researchers to apply neural network-based models to improve IDE efficiency. A recent work trains a neural network that encodes the user's query as the input and predicts the corresponding

J. Li (✉) · Y. Sun · Z. Lei
College of Intelligence and Computing, Tianjin University, Tianjin, China
E-mail: jie.li@tju.edu.cn

S. Chen
School of Data Science, Fudan University, Shanghai, China

G. Andrienko · N. Andrienko
Fraunhofer Institute IAIS, Sankt Augustin, Germany

W. Chen
State Key Lab of Cad &CG, Zhejiang University, Hangzhou, China

Published online: 06 October 2022

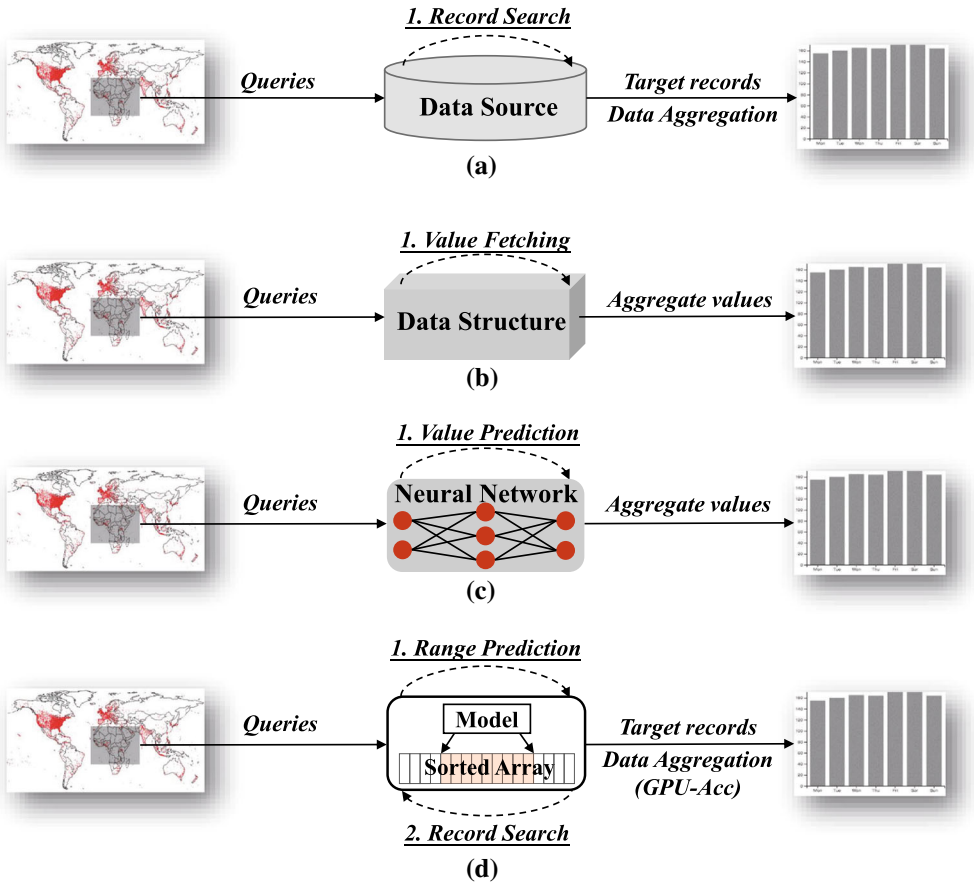



**Fig. 1** Four modes of constructing IDE systems. **a** The traditional method, i.e., searching records satisfying the IDE query and aggregating them for visualization. **b** Data structure-based techniques that pre-store aggregate values of queriable attribute range combinations. **c** Learning-based techniques that encode the user query as input and output aggregate values. (d) Our hybrid approach that first predicts a range and then searches target records within the range

aggregate values (Wang et al. 2021), as in Fig. 1c. The predictive manner avoids preprocessing the data structure. However, the inevitable prediction bias of any machine learning models affects their applicability in scenarios with strict accuracy requirements.

In this paper, we propose a **H**ybrid **P**rediction and **S**earch approach (HPS) to the interactive exploration of big data. HPS follows a two-step workflow that (1) training a regression model to predict a coarse search range for each user-specified query condition and (2) searching target records within the range. We then aggregate these records using GPU-accelerated algorithms to generate visualizations, as in Fig. 1(d). HPS has the advantages of both two types of techniques. The predictive manner avoids constructing the huge data structure storing aggregate values, thus reducing the storage overhead. The search eliminates prediction bias inevitable for any machine learning models.

We test the performance of HPS on multiple open datasets. The feature of returning target records brings better use flexibility. We develop three systems with display/query functions that are unavailable for existing techniques to demonstrate this point, which supports (1) constructing views showing either records or aggregate values, (2) changing the aggregation function during the exploration, and (3) querying on many (e.g., >2) finely binned attributes. Quantitative results show that HPS achieves comparable query speeds to existing techniques on datasets with million-scale records using a single GPU, while its storage cost is much lower than those of existing techniques. Moreover, we can further improve HPS's query speed on larger datasets by adding more hardware resources.

In summary, our main contribution is a hybrid prediction and search approach for building interactive visualization systems of big data, which enables (1) better use flexibility, and (2) much lower storage cost than existing techniques, and (3) continuous improvement in query speed by adding more hardware resources.

## 2 Related work

This section reviews three categories of techniques that support interactive exploration of large datasets.

### 2.1 Structure-based approach

Data Cube is a classic OLAP (Online Analytical Processing) (Chaudhuri and Dayal 1997) technique for processing big data, which allows retrieving aggregate values on any attribute combination in real time. Its main problem is the unsuitability for high-dimensional data, as the combinatorial explosion of attributes results in unacceptable storage overhead.

How to reduce the storage cost becomes the key to the subsequent techniques. Liu et al. (2013) limited the maximum number of indexed attributes of a spatiotemporal dataset to four, based on an observation that spatiotemporal exploration involving more than four attributes is uncommon. Miranda et al. (2018) made use of an implicit temporal hierarchy to construct a memory-efficient index structure. Zhao et al. (2021b) proposed a KD-Tree-based index to support interactive exploration of large time series. Pahins et al. (2019) proposed QDS that supports richer aggregate measures, such as quantiles and cumulative distribution. Ghosh and Eldway (2020) proposed AID* that indexes tiles describing the data distribution rather than original data. Lins et al. (2013) proposed a quadtree-based structure, entitled Nanocubes, to index large spatiotemporal datasets. Nanocubes' branches corresponding to the regions that do not contain any data object are removed to reduce the memory usage. Nanocubes achieve optimized storage costs on sparse datasets and many essential features, such as high-resolution 2D query/display and bias-free outputs. Nanocubes' success makes further reducing index sizes through data structure optimizations without degrading other performance aspects difficult.

As a trade-off between storage overhead and query speed, Liu et al. (2019) utilized a "pre-fetching" strategy to reduce the size of Nanocubes by storing values most likely to be queried in the future. Pahins et al. (2016) propose Hashedcubes that stores beginnings and ends of a list of ordered records instead of aggregate values of attribute combinations. It may, however, not ensure the display accuracy by using a leaf-sized parameter. Mei et al. (2019) design a data representation for querying tabular datasets, that returns only approximate results. Gaining the performance or functionality in one aspect, the above techniques may loose performance in others. Maximally reducing the index size while enabling high-resolution query/display on many finely binned attributes is still a challenge, one of our approach's primary goals.

There are also techniques designed for specific requirements, such as Semantics-space-time Cube (Li et al. 2018b) for exploring large spatiotemporal texts, Gaussian Cubes (Wang et al. 2017) for generating visualization involving complex algorithms, Concave Cubes (Li et al. 2018a) for drawing clustering results, TopKube (Miranda et al. 2017) for showing top-ranked objects, and Trip Cube (Xu et al. 2018) for indexing large trajectories. These techniques, however, are designed for the professional data analysis field, which is different from ours.

### 2.2 Sampling-based approach

Sampling means the selection of a subset of elements to estimate characteristics of the entire dataset. By reducing the amount of data in the calculation, a database engine can return results with a low time latency (Agarwal et al. 2013; Kamat et al. 2014). Kwon et al. (2017) systematically summarized the value of sampling for visual analytics. Sampling-based techniques generate only approximate results. As singular data points may be excluded from the calculation, sampling-based techniques cannot be used in some applications, such as anomaly detection. Moreover, because the calculations are performed on selected samples, the results are uncertain, and an estimate of the uncertainty is needed (Fisher et al. 2012; Chaudhuri et al. 2017).

Progressive exploration is an extension of the sampling approach. This approach calculates results step by step on an incrementally enlarged sample set. Users can thus gradually obtain more accurate results (Hellerstein et al. 1999; Vartak et al. 2015; Haas and Hellerstein 1999). In recent years, progressive exploration has gained attention in human-computer interaction (Fisher et al. 2012) and visualization (Zgraggen et al. 2016). Many visualization systems utilize progressive strategies to improve system response speed when handling large datasets (Li and Ma 2019; Kraska 2021). As a representative progressive IDE technique, Falcon (Moritz et al. 2019) constructs index structures only for the active view (the user is manipulating) to reduce the index size. Research has demonstrated that progressively refined

approximate results are often sufficient for exploratory analysis (Fisher et al. 2012; Crotty et al. 2015; Moritz et al. 2017). Most current works focus on constructing an initial subset of data (Rahman et al. 2017) and how to develop a strategy for tailoring candidate queries (Vartak et al. 2015). Recent studies have analyzed the effectiveness (Zgraggen et al. 2016) and the main challenges (Turkay et al. 2018) of progressive data exploration.

HPS is not a sampling-based technique and does not have the above issues. It enables low storage costs and high-resolution 2D query/display simultaneously. Furthermore, it also enables to show original records by storing records in stored arrays.

### 2.3 Prediction-based approach

There is an increasing trend of applying deep learning in visualization (Xia et al. 2021, 2022; Yuan et al. 2021; Zhao et al. 2021a; Ying et al. 2022; Jie and Chun-qi 2022). Utilizing learning-based techniques in data exploration receives much attention. Rossi et al. (2018) combine visual representations with graph mining and learning techniques to aid in revealing important data insights. Chen et al. (2019) design a deep neural network for lasso selection of 3D point clouds. He et al. (2019) train a deep learning model as a surrogate to enable in-site exploration of the large parameter spaces of ensemble simulations. Chen et al. (2019) utilize the generative model to extract density maps' dynamics by producing interpolation information. Xie et al. (2018) present a hypergraph learning algorithm for visual analytics of heterogeneous datasets. However, these works are different from our approach that aims to form a common framework for exploring various large datasets.

There have been learning-based techniques for interactive data exploration. For example, NeuralCubes (Wang et al. 2021) trains neural networks to approximate datacube. The significant difference between HPS and NeuralCubes (Wang et al. 2021) is that HPS trains models on individual attributes, while NeuralCubes sets a single model to handle all multi-dimensional queries. We propose the structure for two reasons. First, fitting a multi-dimensional distribution is more complicated than a linear one. HPS thus enables a higher prediction accuracy, fewer parameters, and easier model tuning. Second, it avoids massive training sets (HPS's models take single keys as inputs, Sect. 4.2). NeuralCubes (Wang et al. 2021) has to include training samples of attribute range combinations. The number exponentially increases with that of attributes and the binning granularities. NeuralCubes (Wang et al. 2021) thus supports low-resolution queries and returns approximate results only.

Prediction-based techniques have many inherent issues despite small sizes, such as inevitable prediction bias, huge training set, slow training, and lengthy model tuning. HPS avoids them through a series of design optimizations.

## 3 Problem statement

We first define the IDE query and introduce the query decomposition, based on which a first binary search-based IDE solution is given. By carefully analyzing the pros and cons of the solution, we identify HPS's design goal.

### 3.1 IDE query definition

HPS is used to construct IDE systems with multiple coordinated views. Users can select arbitrary value ranges on multiple attributes as a query through common interactions, such as panning, zooming, and brushing. After the user's selection on any view, all other views are updated correspondingly.

Let $D = (a_1, a_2, \ldots, a_n)$ be a dataset with $n$ attributes. Query can be defined as $q = (r_1(a_1), r_2(a_2), \ldots, r_n(a_n))$, where $r_i()$ is an operation that returns a continuous value range on the attribute $a_i$. In all existing techniques, value ranges of attributes are discretized into bins (Lins et al. 2013; Pahins et al. 2016; Liu et al. 2013; Mei et al. 2019), as in Fig. 2. A bin is the smallest unit for value range selection, as in Fig. 2. Each attribute range $r_i(a_i)$ thus is a continuous bin interval.

We can divide value ranges of attributes into an arbitrary number of bins to support either high or low query/display resolution, see the two maps in Fig. 2a. There is no standard for how many bins an attribute range has to be called finely binned. For example, we consider longitudes and latitudes in Nanocubes (Lins et al. 2013) are finely binned, whose value ranges are divided into $2^{25}$ bins, thus enabling the identification

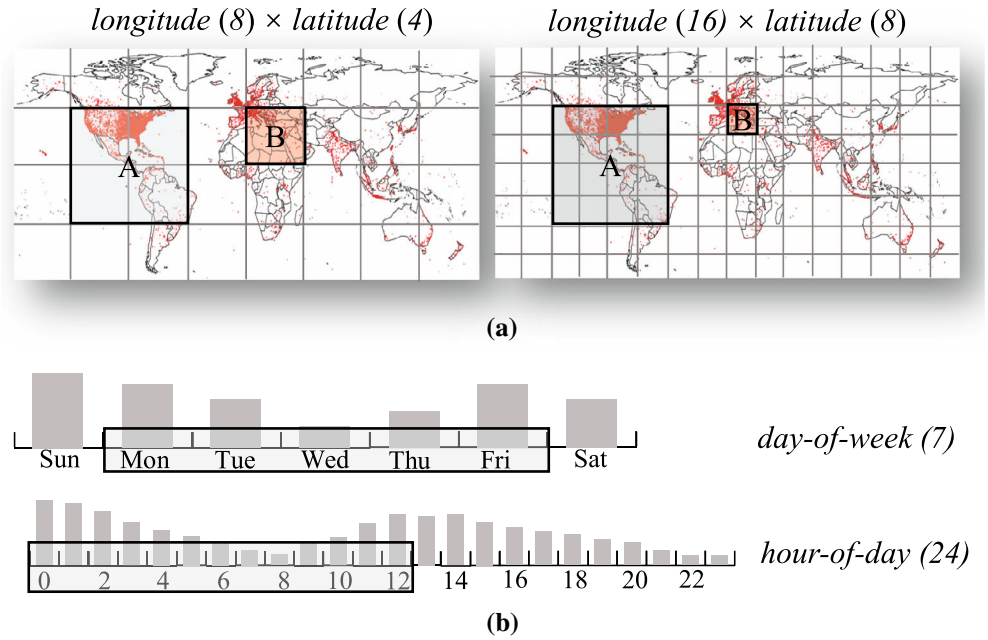



**Fig. 2** Illustration of IDE query. Rectangles with black borders represent the selected attribute ranges, which should cover integral bins. **a** Two binning granularities on longitude and latitude to support different query/display resolutions. **b** Temporal attributes are discretized according to two relevant time cycles

of individual objects in streets. There are also naturally discretized attributes, such as day-of-week and hour-of-day having 7 and 24 bins, respectively, as in Fig. 2b. Their binning granularities are relatively low.

## 3.2 Query decomposition

We can divide a query into many 1D queries, i.e., $q = (q_1, q_2, \ldots, q_n)$, where $q_i = (r_i(a_i))$ is a 1D query with a single specified attribute range. Assume $RS$ is the record set satisfying $q$, $RS_i$ is the record set found by executing $q_i$. It is obvious that $RS = \cap_{i=1}^{n} RS_i$. We thus can get the same records of $q$ by separately executing all 1D queries and calculating the intersection of their results. Figure 3 shows an example of decomposing a spatial query into two 1D queries on longitude and latitude, respectively.

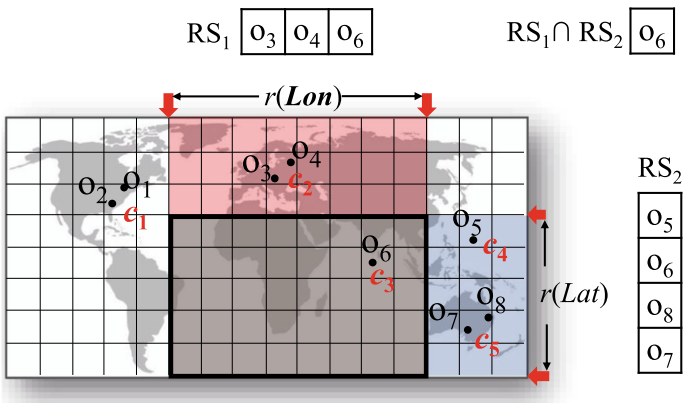



**Fig. 3** Example of the query decomposition. The user brushes a rectangle on a map (marked with a black box) to select objects in it. We decompose the query into two 1D queries specifying value ranges on longitude (marked red) and latitude (marked blue). Two record sets $RS_1$ and $RS_2$ can be found by executing the two 1D queries, while their intersection contains target records within the brushed region

### 3.3 A first search-based solution

We propose a method to find records for each 1D query quickly. As in Fig. 4, we can establish a sorted array for the attribute, which stores all records that are sorted according to their values on the attribute. For each query, we execute the binary search twice to find two positions for the end points of the queried attribute range. Records between the two positions on the sorted array are targets. Because records are sequential in the sorted array, the binary search can be efficient.

Along the line, we can decompose each query into multiple 1D queries and execute all decomposed 1D queries separately. Thus, target records satisfying the IDE query can be quickly obtained by executing all decomposed 1D queries and calculating the intersection of records found in all 1D queries. Figure 5 shows the case of applying the method on a dataset with five attributes.

The advantages of the solution are obvious. First, it can return records satisfying the IDE query. Second, the storage cost is low. We construct a sorted array consisting of all records for each attribute. The overall storage cost is linearly proportional to the number of attributes, unlike most existing techniques, whose sizes will exponentially increase as the number and binning-granularity of attributes to be jointly queried. Third, its outputs are always correct without any bias. In other words, the solution is not a progressive technique that has to sacrifice query precision to achieve a smaller storage cost. Fourth, it adapts to both sparse and dense datasets, unlike most existing techniques designed for sparse datasets exclusively (Lins et al. 2013; Pahins et al. 2016; Mei et al. 2019). Sorted arrays store original records, unrelated to the distribution of records, and we can always find correct records through the binary search. Finally, it applies to frequent data changes. For inserting or deleting an attribute, we simply add or remove the corresponding sorted arrays. For receiving or removing records, we just need to insert (or delete) affected records in sorted arrays, which can be accomplished efficiently, since all records in sorted arrays are sequential.

An overlooked issue is the time overhead of finding records of 1D queries is not fixed and will increase with the number of records. Moreover, the above solution involves on-the-fly calculations, i.e., intersection and aggregation, which are time-consuming when the dataset contains many records. We thus propose HPS to resolve the problems. We will introduce HPS's principle and design optimizations in the next section.

## 4 Approach

We introduce the hybrid approach after a simple description of the principle of predicting the search range. We then give a series of design optimizations to improve the usability of the approach.

### 4.1 Prediction principle

HPS is used to fetch records within the value ranges of 1D queries. Each HPS is coupled with a sorted array to find records within value ranges of an attribute. The sorted array contains all the data records. These records are sorted according to their values on the attribute. The length of the sorted array is equal to the number of records in the dataset. An important observation is that we can draw a curve to reflect the variation in positions along with their attribute values, as in Fig. 6a. The variation is monotonically increasing. Thus, we can set a regression model to fit the curve. For each lookup attribute value $k$ (below we call it a **key** for simplicity), the model can predict the corresponding **position** $p$ on the sorted array, as in Fig. 6b. The attribute value of $p$ is the first one equal to or higher than $k$. The model should run twice to map the two end points of each queried range to two positions. Records between the two positions are what we want to find on the attribute.

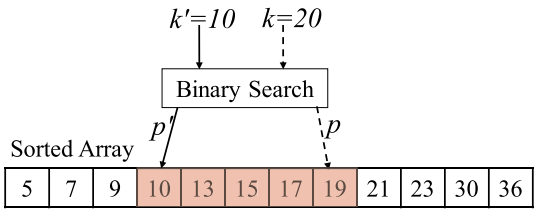



**Fig. 4** Finding records in (10, 20) of an attribute using the binary search. A sorted array is necessary, in which all records are sorted according to their values on the attribute

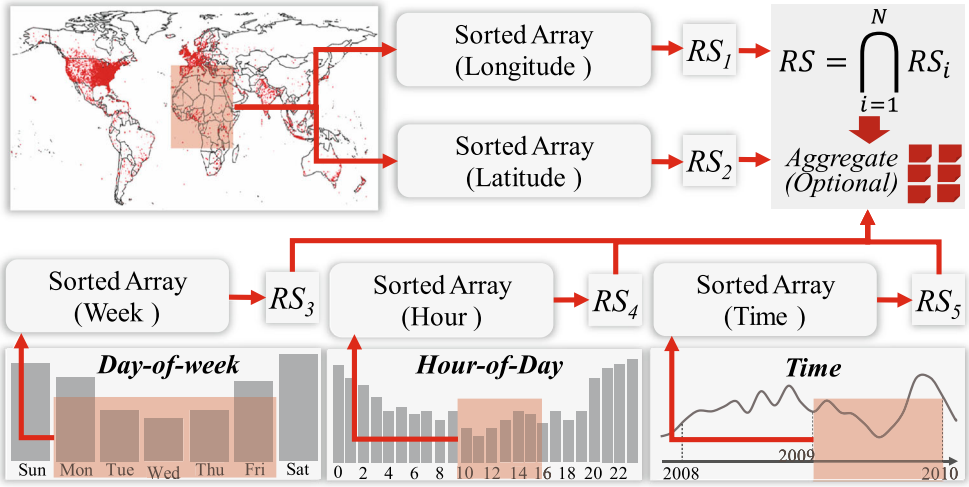



**Fig. 5** IDE of a dataset with five attributes. We create a sorted array for each attribute to find records within the value range of the attribute. Target records can be obtained by calculating the intersection of record sets fetched from different sorted arrays

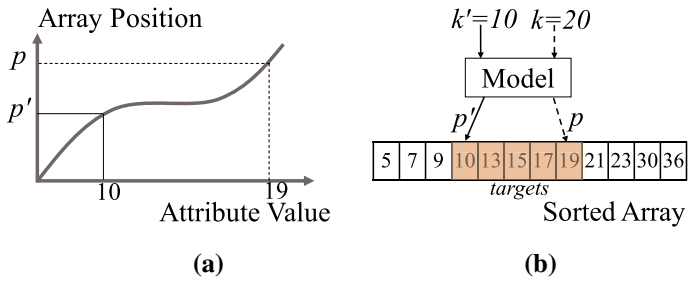



**Fig. 6** Principle of predicting records within an attribute range, depending on two components, i.e., a sorted array and a regression model. **a** The sorted array stores all records. The relationship between positions and their keys on a sorted array is illustrated as a curve. The model is fitting the curve. **b** Finding records within (10, 20). The regression model runs twice to map two end points to the two positions on the sorted array. Records between the two positions are the targets

Training a regression model to fit a monotonically increasing curve is easy. We traverse all positions in a sorted array to collect training samples. Each training sample is in the form of $\langle k, p \rangle$, where $k$ is the queried key and the $p$ is the corresponding position. We include all key-position pairs into the training set. The training set size thus is equal to the number of bins divided on the attribute. For example, the longitude with $2^{25}$ bins results in about 33M training samples for the regression model, which most ordinary GPU servers can handle.

For an attribute with fewer bins, such as day-of-week (7) and hour-of-day (24) (see Fig. 5), we design a hash-map storing all key-position pairs to support the query instead of training a model. For example, hash-maps of day-of-week and hour-of-day contain 7 and 24 key-position pairs, respectively. The hash-map can return the true position immediately without any bias for a key. This design aims at improving efficiency by skipping unnecessary training processes. Moreover, it can reduce the storage cost, since a hash-map with a few pairs is much smaller than a model.

## 4.2 Hybrid prediction and search

Regression models are good at learning a high-level trend. The prediction result can be very close to the true value, but it is impossible to eliminate the small biases, as in Fig. 7a, affecting the correctness of the generated views. We utilize a hybrid prediction and search method to find the true position of a key. First, we use the model to predict a position $p$. Second, we set an initial search range threshold *thres* and use binary search to find the true position (the first position with the attribute value equal or larger than the queried key) within $[p - thres, p + thres]$. **If the true position is not in the interval, we double** *thres* **and**

**search again. This process repeats until finding the true position.** The binary search can be very efficient. Figure 7b shows a case of finding the position of key 24.

In addition to avoiding the storage of aggregate values, the prediction shortens the search range, making the search time a constant (Sect. 6.5). The method also reduces the training time, because we can stop training whenever the current bias is less than the initial search range (we actually implement a regression model as a number of small models, which will be introduced in the next section, and the method is also suitable for each small model to reduce the training time). The effects of the initial search range *thres* on training time are tested and summarized in Table 2.

## 4.3 Design optimizations

We propose the following optimizations to resolve prediction-related issues, which better adapt the HPS to real-world scenarios.

### *4.3.1 GPU-based parallel computation framework*

**Challenge 1** *HPS involves on-the-fly Intersection and aggregation, which are time-consuming when the data volume is huge. Improving their efficiency via algorithm optimizations is difficult.*

We design the GPU-based **P**arallel **C**omputation **F**ramework (PCF) to accelerate these operations, which (1) adopts a divide-and-conquer strategy, partitioning each computational operation into multiple independent sub-operations, and starts multiple threads to execute these sub-operations in parallel; (2) uses bitmaps to represent query results, enabling to execute intersection and aggregation via simple bitwise bit operations, faster than traditional set operations; (3) automatically assigns sub-operations to multiple GPUs, further increasing concurrent threads. Specifically, PCF involves three steps, as in Fig. 8 and detailed below:

**Bitmap generation** PCF creates a bitmap (Chan and Ioannidis 1998) with width equal to the number of all records and utilizes a bitmap to encode query results of each 1D query by setting bits corresponding to the found records to 1 and others to 0, as in Fig. 8a.

**Bitwise AND-based intersection** PCF divides each bitmap into multiple parts and execute bitwise AND operation to get the intersection of each pair of shorter bitmaps in a thread, as in Fig. 8b. Bitwise AND operations on different threads can be executed in parallel. This step forms a single bitmap encoding target records of the IDE query.

**Aggregation** PCF again divides the single bitmap encoding the target records into multiple parts and conducts aggregation on each part in a thread, as in Fig. 8c. Aggregation results of different threads can be further aggregated using the same multi-thread method until obtaining desired aggregate values.

### *4.3.2 Segmented curve fitting: implement a regression model as many parallel-aligned small models*

**Challenge 2** *The key-position curve of a sorted array always shows a nonlinear and fluctuating trend (although monotonic increasing). Fitting such a curve with a single model always causes large prediction biases and requires more model parameters and tedious tuning* (Shazeer et al. 2017), as in Fig. 9a.

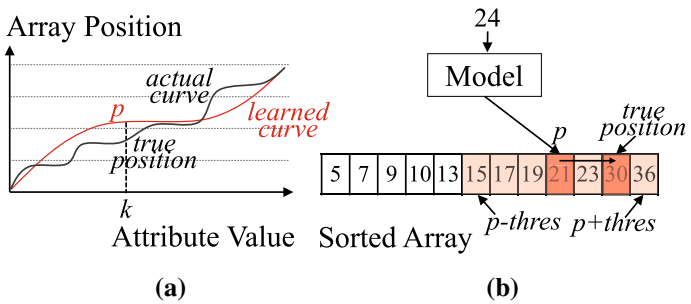



**Fig. 7** Hybrid prediction and search. **a** The prediction bias is inevitable even for a model well capturing the general trend of the curve. **b** An example of predicting the position of 24. Having obtained a predicted position $P$, HPS searches for the true position within $[p-3, p+3]$, where 3 is the preset initial search range threshold

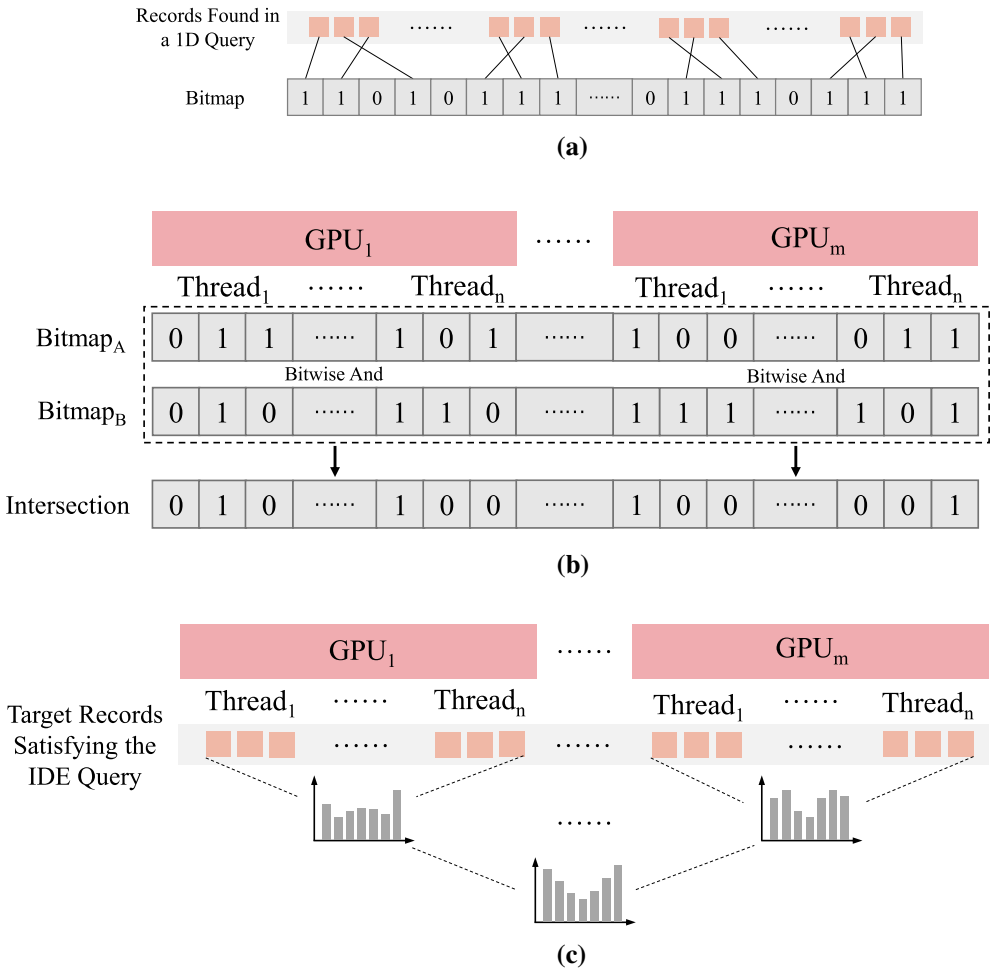



**Fig. 8** Three steps of PCF, i.e., **a** bitmap generation, **b** bitwise AND-based intersection, and **c** aggregation

**Challenge 3** *The single model structure is not efficient in the model update. Once the data are updated, the whole model may need to be retrained with all training samples.*

We thus propose segmented curve fitting (SCF) that divides an entire value-position curve into a large number of segments and set a group of small models (denoted as $NN_i$), each fitting a segment of the data curve, as in Fig. 9b. As the number of divided segments increases, the trend of each segment gradually approximates a straight line. Thus, each small model can have a simple structure with few parameters, which reduces the difficulty of model tuning. The simple structure also enables frequent data update scenario in IDE. For the data change affecting the curve shape, only small models of the affected segments need to be retrained. Thus, the cost of retraining models is significantly reduced.

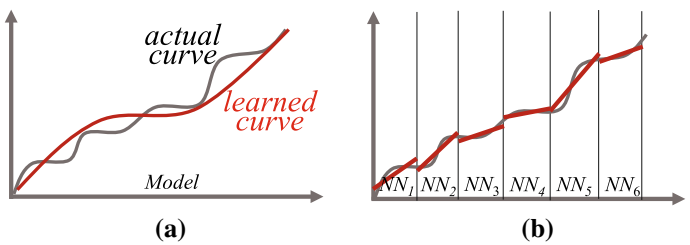



**Fig. 9** The basic idea of the segmented curve fitting. **a** Fitting a key-position curve with a single model always causes large prediction biases. **b** We divide the curve into multiple segments and set a small model to fit each segment to improve the prediction accuracy

Figure 10a shows the SCF method. SCF integrates many small models ($NN_1$-$NN_k$). Each small model works on one segment. As in Fig. 10b, $(s_1, s_2)$ is for $NN_1$, $(s_2, s_3)$ is for $NN_2$, $(s_3, s_4)$ is for $NN_3$, and so on. There is a model selector. The model selector assigns a key to a small model for predicting its array position according to value of the key (i.e., the key is assigned to the small model whose segment covers the attribute value of the key). For example, in Fig. 10a, the queried value is within $(s_2, s_3)$, thus $NN_2$ is picked and the predicted position is between $(p_1, p_2)$. The model selector is only a thread that keeps all split points, i.e., $s_1$-$s_4$.

As in Fig. 10b, we make all segments to cover equal value ranges, i.e., $(s_1, s_2)$, $(s_2, s_3)$, $(s_3, s_4)$,..., have the same width (bin interval). Thus, each segment has the same number of bins, making the number of keys (bin indexes) assigned to each small model equal. We can thus use a unified structure for all small models. Moreover, when a small model completes the training session, its weights can be used to initialize the next small model, and so forth. This relaying process accelerates the entire training.

We need to divide the curve into a sufficient number of segments, such that each segment can be approximated as a straight line. Simple linear regression can then be used to capture the linear trend with high accuracy. The "number of small models" is a parameter that will affect the training time and prediction accuracy. We test these effects in an experiment (Sect. 6.6).

Each small model has a "buffer" (see pink rectangles in Fig. 10a) and a "variable" (see yellow rectangles in Fig. 10a) to facilitate model updates. We will introduce their usages and the algorithm to the update of the small models in supplementary material.

SCF has many benefits: (1) Each small model learns a segment of the trend. As the number of small models increases, trends in individual segments become more linear, resulting in higher predicting accuracy. (2) Each small model has a simple structure to fit a short segment. Running such a small model for a query consumes few resources, such as GPU time and memory. (3) For any data change, only the small models affected by the change may need to be retrained, thus avoiding retraining the entire structure and ensuring high update efficiency.

#### *4.3.3 Replace records with cells in sorted arrays*

**Challenge 4** *HPS has two types of storage cost, i.e., regression models and sorted arrays. When the dataset contains a large number of records, the sizes of sorted arrays are still huge.*

We thus store cells that contain at least one record instead of original data records in sorted arrays. A cell has 1-bin width on each attribute. In Fig. 11, $c_1 - c_5$ are five 2D cells. Because many records are in the same cells, the number of cells is much less than the original records, thus taking up less space. This design works for both sparse and dense datasets. For sparse datasets, because records are in fewer cells, more space can be saved. Figure 11 shows a case of storing five cells rather than eight records in sorted arrays.

We need to calculate one or more cell attributes for each cell based on its inner records according to what the views will show. For example, if the view shows "Count" patterns, we need to calculate the number of records in each cell as a cell attribute. We can get the measure values by performing algebraic operations based on the selected cells. Note that, replacing records with cells is optional, which does not affect the

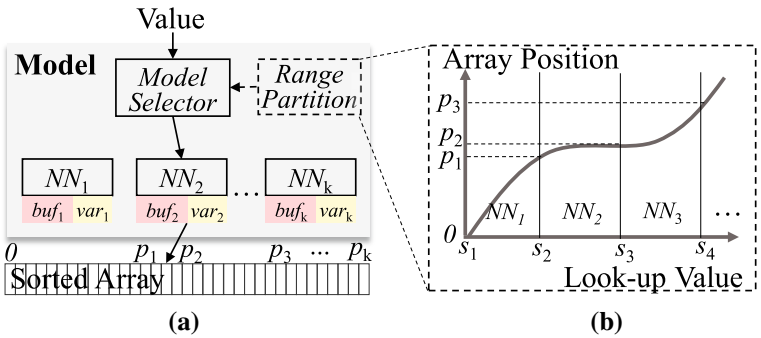



**Fig. 10** Segmented curve fitting. **a** A model is implemented as a set of parallel-aligned small models. At the first level, the model selector assigns a key to a small model for predicting the position. **b** The value range of an attribute is equally divided into multiple segments, each containing an equal number of bins. Keys whose attribute values are in a segment are assigned to the corresponding small model

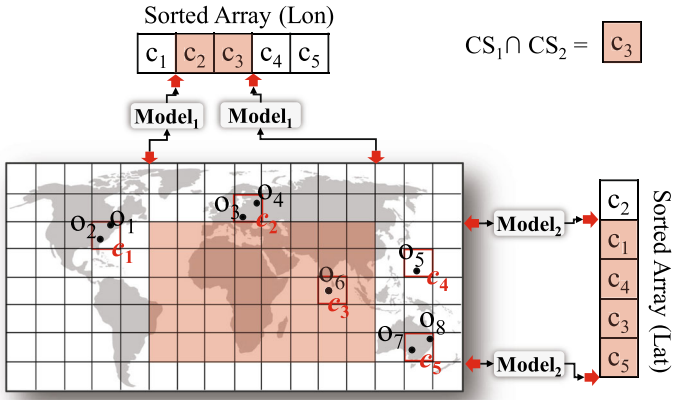



**Fig. 11** Replacing eight records with five cells in sorted arrays to save space. We thus run each model twice to find a set of cells rather than original records. $CS_1$ and $CS_2$ are two cell sets found by executing the two 1D queries

display/query effects because a cell is the smallest unit for query and display. Records in a cell will overlap and should be selected or unselected together in views.

## 5 Implementation

Each small model fits a segment of a curve with an approximately linear trend. We thus implement each small model as a neural network without any hidden layer, which is equivalent to a linear regression model. We can directly use linear regression or other classic machine learning techniques to achieve the same effects. We choose neural networks since developers can conveniently add a hidden layer where the curve has an extremely irregular trend. In most case, a dozen of neurons per layer is sufficient to ensure high accuracy, while training such a neural network takes only a few seconds (or $<1s$). Even with hundreds of thousands of neural networks, the overall storage cost and training time are still acceptable.

The MSE (Mean Squared Error) loss and ReLu activation function are used. Optimization strategies, such as learning rate decay and dropout, are optional. The initial search range threshold is set to 512 by default. The training of each small model can be terminated early when the average prediction bias is less than the threshold to accelerate the SCF building. In general, all used parameters are standard, integrated into existing deep learning frameworks.

HPS has a learning-based structure,which always has tens of thousands of small models and depends on many running-time libraries, such as pytorch, and CType. This makes integrating HPS with visualization systems difficult. One can use some open source frameworks, such as Django, to alleviate this problem. We also plan to encapsulate the query and training interfaces to simplify the applications.

## 6 Evaluation

The evaluation consists of three parts. First, we use HPS to develop visualization systems with different display/query functions (Sect. 6.2). Second, we test the storage usage, training time and execution speed of HPS and compare HPS's measures to those of Nanocubes (Sects. 6.3–6.5). Finally, we analyze the impacts of the parameter, i.e., the number of small models in a single regression model, on HPS's performance (Sect. 6.6). All the experiments are conducted on a GPU Server (XEON E5-2680, 196G, 2080Ti×4).

### 6.1 Experiment preparation

*Dataset* We collect six real-world open datasets and one synthetic dataset in evaluation, including (1) brightkite social media checkins (Cho et al. 2011), (2–3) New York taxi trajectories (yellow and green), (4) Chicago crime records, (5) gowalla social media dataset, (6) US flight on-time statistics, and (7) ScatterPlot matrix (SPLOM) synthetic dataset. Table 1 summarizes the details of the first six datasets. The SPLOM

**Table 1** Dataset information and the storage cost summary

| | Records(N) | Interaction schema ($2^N$) | Model(N) | Cells(N) | ModelSize | ArraySize | Σ |
|---|---|---|---|---|---|---|---|
| Brightkite | 4.5M | lon(25), lat(25), day-of-week(3), hour-of-day(5), time(16) | 2 | 3.5M | 6.1 MB | 66.5 MB | 72.6 MB |
| taxi-yellow | 258.1M | lon1(25), lat1(25), lon2(25), lat2(25), day-of-week(3), hour-of-day(5), time(10) | 4 | 249.3M | 12.2 MB | 6.5 GB | 6.5 GB |
| taxi-green | 28.3M | lon1(25), lat1(25), lon2(25), lat2(25), day-of-week(3), hour-of-day(5), time(10) | 4 | 28.1M | 12.2 MB | 751.1 MB | 763.3 MB |
| crime | 6.8M | lon(25), lat(25), type(6), day-of-week(3), hour-of-day(5), time(13) | 2 | 6.6M | 6.1 MB | 152.2 MB | 158.3 MB |
| gowalla | 6.4M | lon(25), lat(25), hour-of-day (5), day-of-week (3), time (15) | 2 | 6.1M | 6.1 MB | 116.9 MB | 123.0 MB |
| flight | 123.5M | lon(25), lat(25), time(16), carrier(5), takeoff-delay(4) | 2 | 43.5M | 6.1 MB | 829.0 MB | 835.1 MB |

dataset is synthesized according to the setting of imMens Liu et al. (2013). Among the six real-world datasets, (2), (3) and (4) have relatively dense distributions, while the other three are sparse.

*Attribute selection* We select 5–7 attributes for each dataset, as in Table 1. Four datasets (brightkite, crime, gowalla and flight) have the same attributes as Nanocubes Lins et al. (2013), while those of other two datasets (taxi-yellow and -green) are the same as Hashedcubes Pahins et al. (2016).

*Discretization* We uniformly divide longitudes and latitudes of all the six datasets into $2^{25}$ bins. Temporal attributes are discretized according to the natural division of time. Other attributes have the same binning granularities as Lins et al. (2013) or Pahins et al. (2016), as in Table 1.

*Model and sorted array* We train models for longitudes and latitudes. These attributes are all divided into $2^{25}$ bins. Other temporal attributes use hash-maps. A sorted array is created for each attribute. We store cells rather than original records in a sorted array.

*Collection of training samples* We collect all key-value pairs as the training samples of a model. Since longitudes and latitudes are divided into $2^{25}$ bins, each model has about 33M training samples.

*The number of small models* According to the experimental results, we uniformly set 20K small models for longitudes and latitudes of all the datasets, which can achieve relatively good performance in most cases (Sect. 6.6). Each small model thus has about 1677 training samples ($2^{25}/20K$).

*Training setting* Each small model is implemented as a neural network without hidden layer. All the small models have the same structure: a $1 \times 20$ input layer and a $20 \times 1$ output layer, 40 weights in total. The MSE (Mean Squared Error) loss is used. The learning rate is set to 0.05, and the batch size is set to 16. The learning rate decay is used to change the learning rate to 0.7 times of the last one every five epochs. The initial search range threshold is set to 512. The training stops whenever its epoch number reaches 120, or the average actual prediction bias is less than 512.

## 6.2 Visualization systems

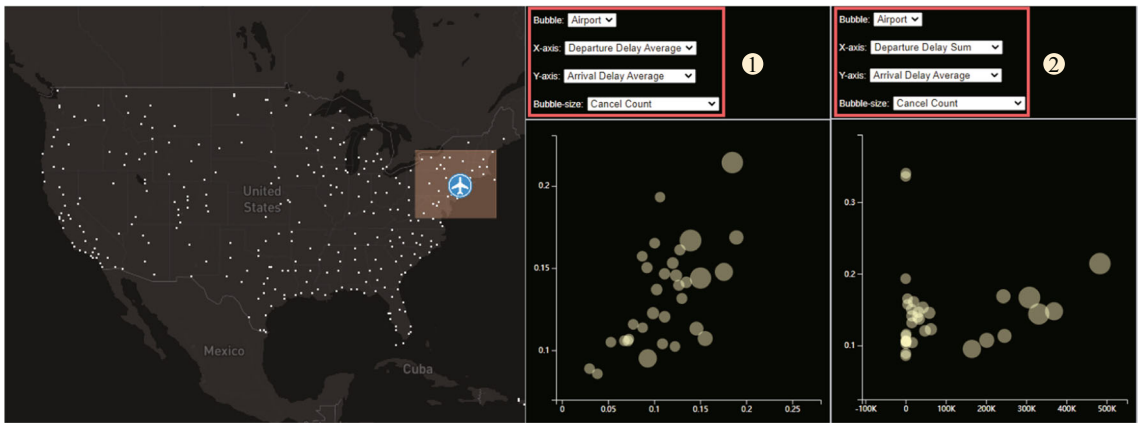
We develop three visualization systems, each with a unique display/query function, on three representative real-world datasets to demonstrate HPS's better use flexibility, as follows:

*Showing both aggregate patterns and original records* We constructed a system on the Brightkite dataset. It contains a map and four views showing temporal patterns, as in Fig. 12a. The map can show aggregate spatial patterns (Fig. 12(a1)) or filtered records' distribution (Fig. 12(a2)).

*Changing aggregate measures interactively* We develop a system on the US flight dataset that records on-time statistics. As in Fig. 12b, the system includes a map showing airports' positions and a scatter-plot showing the flight delay situations. Notably, users can interactively change what scatters and axes map in the scatter-plot. For example, each scatter can be an airport, a state, or an airline; the scatter size and the two axes can map different aggregate measures, such as the count of departures and the average of arrival delays.

*Querying on more finely binned attributes* We develop a visualization system to enable O-D analysis on the yellow-taxi dataset with 258.1M records. The interface is shown in Fig. 13, which has two maps for


**(a)**




**(b)**

**Fig. 12** Two IDE systems developed using HPS. (a1) A system integrates five views showing aggregate patterns of the brightkite dataset. The map in (a1) can be switched to show the filtered records' positions directly (a2). **b** A system for analyzing the US flight dataset, in which users can interactively change the mappings of scatters and two axes in the scatter-plot during the exploration

users to select origins and destinations of taxi trips, and four small views showing temporal patterns. Latitudes and longitudes of the two maps are uniformly divided into $2^{25}$ bins. That is users can query on four finely binned attributes at the same time.

## 6.3 Memory usage

We record the storage costs of HPS constructed on six open datasets. Table 1 shows the dataset information and the storage costs of HPS on different datasets. The number of data records and the interaction schema are in the first two columns. "Model(N)" indicates the numbers of small models. "Cells(N)" represents the number of cells in a sorted array. "ModelSize" and "ArraySize" indicate the sizes of all models and sorted arrays, respectively. "Σ" represents the overall storage cost, i.e., the sum of "ModelSize" and "ArraySize."


**Fig. 13** A system for analyzing O-D patterns of the taxi-yellow dataset. Longitudes and latitudes of the two maps are uniformly divided into $2^{25}$ bins to enable high-resolution query/display

In general, the storage cost of HPS is low. A model is about several MBs. Sorted arrays take up most of the storage, ranging from tens of MBs to several GBs, depending on (1) the number of cells, (2) the number of attributes, and (3) the data distribution. Specifically, more records result in more cells, and the number of attributes determines that of sorted arrays to be stored, which explains why taxi-yellow, taxi-green, and flight datasets have significantly higher storage costs than those of the other three datasets. Moreover, data distribution slightly affects this aspect. Two datasets with similar numbers of records may have different storage costs, since their records can be in different numbers of cells, see the crime and gowalla datasets in Table 1.

Figure 14a compares the sizes of HPS and Nanocubes Lins et al. (2013) on the four datasets which are representative with varying scales of records. As expected, HPS are significantly smaller on all the datasets. Most existing structure-based techniques are designed to store aggregate values of attribute combinations, unnecessary for HPS.

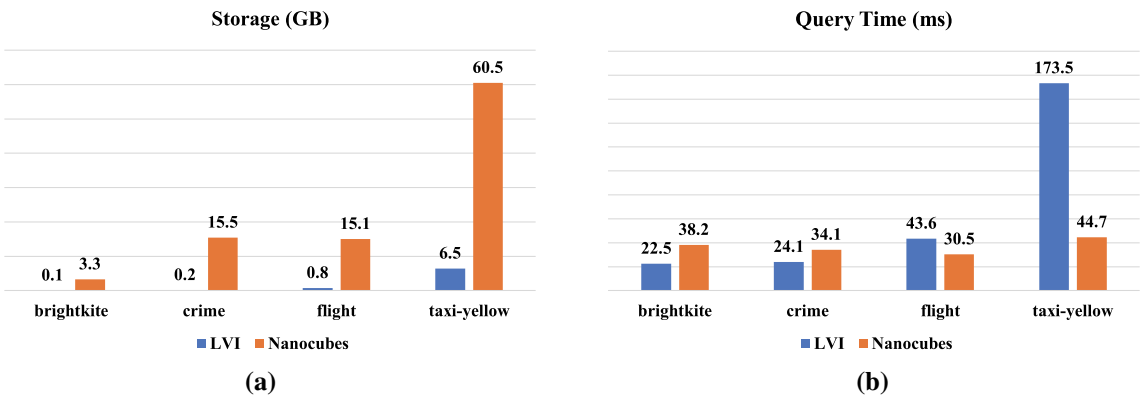



**Fig. 14** Comparison of HPS with Nanocubes on **a** storage cost and **b** execution time. HPS is significantly lower than Nanocubes in storage and has a faster execution speed on million-scale datasets. However, as the number of records increases, the query speed of HPS will decrease. We use Nanocubes of V4, larger than V1 used in Lins et al. (2013) since V4 treats different types of attributes (spatial, temporal, categorical, numerical) uniformly. We test the execution time using a single-GPU condition

**Table 2** Training time of a single small model under different initial search range thresholds. We use 512 by default (marked with bold font) for other tests

| | 1024 (>*thres*) | **512** (>*thres*) | 256 (>*thres*) | 128 (>*thres*) |
|---|---|---|---|---|
| Brightkite | 1.5h (0.30%) | **1.9h (0.91%)** | 2.6h (1.36%) | 3.8h (1.54%) |
| Taxi-yellow | 1.6h (0.16%) | **1.7h (0.19%)** | 1.8h (0.21%) | 1.8h (0.26%) |
| Taxi-green | 1.7h (0.16%) | **1.7h (0.17%)** | 1.7h (0.18%) | 1.7h (0.19%) |
| Crime | 1.5h (0.18%) | **1.5h (0.20%)** | 1.5h (0.21%) | 1.5h (0.21%) |
| Gowalla | 1.5h (0.39%) | **2.1h (1.11%)** | 2.9h (1.59%) | 4.0h (1.83%) |
| Flight | 2.1h (1.28%) | **2.5h (1.35%)** | 2.5h (1.42%) | 2.6h (1.46%) |

Numbers in parentheses are percentages of keys with biases larger than the threshold

### 6.4 Training time

Table 2 shows the training time of a single SCF. We set different initial search range thresholds (training is terminated when the current average bias is less than the threshold, Sect. 4.3.2) and observe the training time variations.

Table 2 shows the experiment results. We find that SCF's training is fast. Specifically, the training time is between 1 and 3 h when using the default initial search range threshold 512, as many training sessions can be terminated quickly, no need to execute all the epochs. Thus, we can set a larger threshold to reduce the training time further.

The percentages of keys with prediction biases larger than the initial search range threshold are low (see numbers in parentheses in Table 2), implying high overall accuracy. Like the training time, the percentage is also inversely proportional to the initial search range threshold, which can be further reduced by setting a larger search range threshold.

### 6.5 Execution speed

We test the execution time of PCF with different numbers of GPUs. We find that the execution of HPS can be further improved by using more GPUs. See supplementary materials for details.

We then extract 10,000 actual user queries from logs of the visualization systems and test the execution time of HPS in real-world scenarios by running them. We require all the selected attribute ranges to cover hot places of different zoom levels, such as countries, cities, and blocks, and diverse periods. The task is the same as RSATree Mei et al. (2019).

We test the query speed using a single GPU. The execution time has three parts, i.e., predicting and searching positions, copying cells from main memory to GPU, and calculating intersection and aggregation, as in Table 3. The total time overhead also includes a certain amount of network latency. In general, the total time is short, supporting our claim, i.e., HPS supports interactive data exploration (Liu and Heer 2014). Specifically, prediction and search take almost a constant time, while the other two take longer when datasets contain more cells. Besides, we test the search speed using Binary search, as in Table 3.

We also establish Nanocubes on four different datasets to compare Nanocubes' execution speed with HPS. We deploy all the established Nanocubes in the GPU server and execute the same 10,000 queries on Nanocubes (without GPU Acceleration) and HPS (using a single GPU). The result is shown in Fig. 14b. In general, HPS executes a little faster than Nanocubes on million-scale datasets. However, HPS is slower than Nanocubes on larger datasets. Moreover, the speed of Nanocubes is almost constant on different datasets. In contrast, HPS involves on-the-fly calculations, resulting in varying query time on datasets of different scales.

We finally test HPS's query speed under different numbers of GPUs (using the same 10,000 queries). Figure 15 shows the results. We find the average query time can greatly decrease by using multiple GPUs for larger datasets, i.e., taxi-yellow, taxi-green, and flight, illustrating HPS's better scalability. However, the other three million-scale datasets' query time, i.e., brightkite, gowalla, and crime, increase slightly. This result also supports the conclusion we made about the PCF, that using a single GPU already makes the computation of millions of records fully parallel while adding GPUs will bring additional time overhead in device scheduling and synchronization.

**Table 3** Execution time of HPS consisting of three parts: position prediction and search, copy cells from main memory to GPU, and calculation of intersection and aggregation (at initial search range threshold of 512)

| | | Prediction (μs) | Binary search (ms) | Copy (ms) | Calculation (ms) | Total (ms) |
|---|---|---|---|---|---|---|
| Brightkite | Mean | 1.2 | 213.1 | 1.0 | 10.9 | 22.5 |
| | Max | 10.0 | 258.1 | 4.0 | 19.1 | 36.1 |
| | Std | 3.8 | 8.4 | 1.1 | 2.8 | 4.5 |
| | Median | 1.4 | 210.2 | 0.8 | 11.0 | 22.7 |
| Taxi-yellow | Mean | 0.8 | 211.0 | 11.7 | 146.8 | 173.5 |
| | Max | 9.9 | 262.4 | 287.2 | 296.0 | 335.0 |
| | Std | 3.2 | 8.6 | 4.9 | 17.4 | 23.8 |
| | Median | 1.1 | 208.0 | 12.0 | 145.5 | 172.9 |
| Taxi-green | Mean | 1.3 | 214.3 | 2.8 | 30.4 | 46.8 |
| | Max | 10.6 | 285.1 | 7.0 | 46.9 | 65.4 |
| | Std | 4.6 | 8.4 | 1.9 | 3.9 | 5.3 |
| | Median | 1.4 | 210.3 | 2.6 | 30.0 | 46.6 |
| Crime | Mean | 0.8 | 211.2 | 1.2 | 12.2 | 24.1 |
| | Max | 10.0 | 266.5 | 4.7 | 25.0 | 43.0 |
| | Std | 2.7 | 8.6 | 1.4 | 3.1 | 3.9 |
| | Median | 0.8 | 208.5 | 1.1 | 12.2 | 24.3 |
| Gowalla | Mean | 1.8 | 215.4 | 1.4 | 12.4 | 24.8 |
| | Max | 18.0 | 303.8 | 5.1 | 29.1 | 45.2 |
| | Std | 4.0 | 15.3 | 2.0 | 3.5 | 4.1 |
| | Median | 1.5 | 210.4 | 1.5 | 14.8 | 26.0 |
| Flight | Mean | 1.4 | 214.7 | 4.1 | 30.8 | 43.6 |
| | Max | 10.0 | 279.5 | 18.2 | 42.9 | 72.2 |
| | Std | 3.5 | 9.98 | 2.4 | 3.2 | 3.9 |
| | Median | 0.9 | 211.8 | 11.9 | 31.0 | 48.5 |

Besides, execution time of binary search

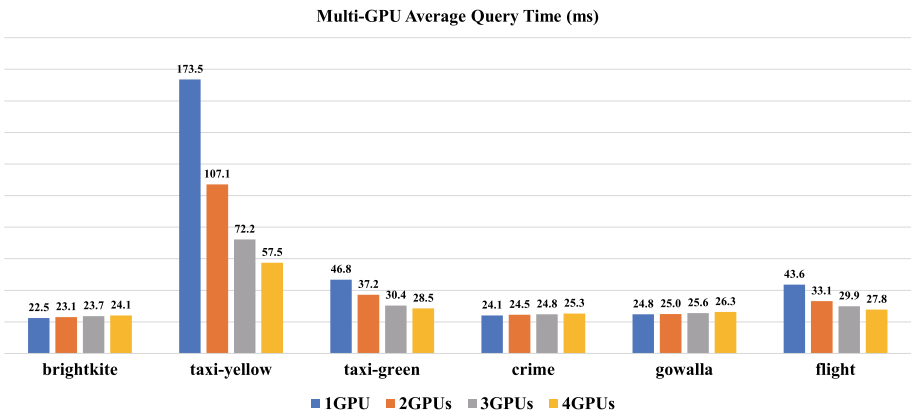



**Fig. 15** The average query time of the six open datasets when using different numbers of GPUs

### 6.6 Parameter setting

Our model involves a hyperparameter, i.e., the number of small models in a single SCF. In order to help users set this parameter, we conduct a quantitative experiment. The results show that the difference in training time of different numbers of small models is not large. It is feasible to set tens of thousands of small models for tens of millions of queriable keys. Details can be found in the supplementary material.

## 7 Limitation analysis

We now discuss the limitations of HPS and the effects on applying HPS in real-world scenarios, as follows:

*Query speed deterioration* As the number of records increases, the response speed of queries will decrease, interactive operations may become less smooth. Although using more GPUs can mitigate this problem, it cannot change the fact that HPS is not a technique with $O(1)$ query speed. A good aspect is that

HPS can achieve comparable performance to existing techniques on million-scale datasets on a single GPU, while the query speed can be further improved by using more GPUs, illustrating HPS's usability in most common scenarios. However, HPS's query speed is not always slow, even for an extremely huge dataset. The response time of a query depends on the number of selected records. Users always explore local patterns (e.g., specific cities, blocks, hour-of-days, day-of-weeks, etc.) after seeing an initial overview. The frequency of selecting large attribute ranges, such as the entire map or period, to include all records is relatively low. Therefore, most queries in visualization systems involve parts of records and are fast to execute.

*GPU dependence* HPS depends on GPU to train models and accelerate calculations. It is a trend for IDE techniques to utilize GPUs. For example, Falcon Moritz et al. (2019) depends on a GPU-based database to accelerate the calculation of aggregate values. Of course, we can implement and apply HPS in devices without GPU when hard real-time is not necessary. We can exclude all the models and find correct positions of queried attribute values through the binary search. The search time thus depends on the number of records. We can also use CPU-based multi-thread algorithms to accelerate the intersection and aggregation.

*Requiring model training expertise* HPS requires developers to know neural networks (or linear regression) to set parameters, such as learning rate, number of iterations, and activation function. However, HPS only uses ordinary neural networks without hidden layer. The simple structure reduces the difficulty of model tuning. In most cases, users can achieve good performance through a few trials. Besides, the parameters we have given are reusable in most common cases, although not always optimal.

*Query continuity* Our approach requires each query to cover a continuous spatial or temporal range. However, for a query covering multiple separate value ranges, we can simply split the query into multiple with continuous value ranges on the attribute.

## 8 Conclusion and future work

This paper has presented a novel IDE technique. Unlike existing techniques using either pre-storage or learning-based methods, our approach follows a hybrid strategy to obtain the advantages of both types of techniques, which, according to our knowledge, is the first attempt in visualization. Our approach has smaller storage overheads and better use flexibility than existing techniques but involves intersection and aggregation for each query. We thus design a GPU-based framework to accelerate these on-the-fly operations. Our approach is not for improving existing techniques. Instead, its technical characteristics are suitable for scenarios with special hardware conditions and visualization requirements, such as low memory devices and querying on multiple high-resolution views. Our approach brings a new research idea, which may promote the emergence of more works that combine traditional data structures and learning models for efficient IDE.

In the future, we plan to make two improvements. First, we will further test the scalability of HPS on larger datasets. Second, we will use other AI techniques (e.g., CNN, RNN, and GNN) to support more complex visualization, such as trees and graphs.

**Acknowledgements** This work is supported by the NSFC Project (61972278) and Natural Science Foundation of Tianjin (20JCQNJC01620).

## References

Agarwal S, Mozafari B, Panda A, Milner H, Madden S, Stoica I (2013) Blinkdb: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM European conference on computer systems, pp 29–42. ACM

Chan C-Y, Ioannidis YE (1998) Bitmap index design and evaluation. In: Proceedings of the 1998 ACM SIGMOD international conference on management of data, pp 355–366

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

Chaudhuri S, Ding B, Kandula S (2017) Approximate query processing: no silver bullet. In: Proceedings of the 2017 ACM International Conference on Management of Data, pp 511–519

Chen Z, Zeng W, Yang Z, Yu L, Fu C-W, Qu H (2019) Lassonet: deep lasso-selection of 3d point clouds. IEEE Trans Vis Comput Graph 26(1):195–204

Chen C, Wang C, Bai X, Zhang P, Li C (2019) Generativemap: visualization and exploration of dynamic density maps via generative learning model. IEEE Trans Vis Comput Graph 26(1):216–226

Cho E, Myers SA, Leskovec J (2011) Friendship and mobility: user movement in location-based social networks. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining, pp 1082–1090. ACM
Crotty A, Galakatos A, Zgraggen E, Binnig C, Kraska T (2015) Vizdom: interactive analytics through pen and touch. Proc VLDB Endow 8(12):2024–2027
Fisher D, Popov I, Drucker S, et al (2012) Trust me, I'm partially right: incremental visualization lets analysts explore large datasets faster. In: Proceedings of the SIGCHI conference on human factors in computing systems, pp 1673–1682. ACM
Ghosh S, Eldway A (2020) Aid*: a spatial index for visual exploration of geo-spatial data. IEEE Trans Knowl Data Eng 34(8):3569–3582. https://doi.org/10.1109/TKDE.2020.3026657
Haas PJ, Hellerstein JM (1999) Ripple joins for online aggregation. ACM SIGMOD Rec 28(2):287–298
He W, Wang J, Guo H, Wang K-C, Shen H-W, Raj M, Nashed YS, Peterka T (2019) Insitunet: deep image synthesis for parameter space exploration of ensemble simulations. IEEE Trans Vis Comput Graph 26(1):23–33
Hellerstein JM, Avnur R, Chou A, Hidber C, Olston C, Raman V, Roth T, Haas PJ (1999) Interactive data analysis: the control project. Computer 32(8):51–59
Jie L, Chun-qi Z (2022) Incorporation of human knowledge into data embeddings to improve pattern significance and interpretability. In: 2022 IEEE visualization conference (VIS). https://doi.org/10.1109/TVCG.2022.3209382
Kamat N, Jayachandran P, Tunga K, Nandi A (2014) Distributed and interactive cube exploration. In: 2014 IEEE 30th international conference on data engineering, pp 472–483. IEEE
Kraska T (2021) Northstar: An interactive data science system [J]. VLDB Endowment
Kwon BC, Verma J, Haas PJ, Demiralp C (2017) Sampling for scalable visual analytics. IEEE Comput Graph Appl 37(1):100–108
Li JK, Ma K-L (2019) P5: portable progressive parallel processing pipelines for interactive data analysis and visualization. IEEE Trans Vis Comput Graph 26(1):1151–1160
Li M, Choudhury FM, Bao Z, Samet H, Sellis T (2018a) Concavecubes: supporting cluster-based geographical visualization in large data scale. Comput Graph Forum 37(3):217–228
Li J, Chen S, Zhang K, Andrienko G, Andrienko N (2018b) COPE: interactive exploration of co-occurrence patterns in spatial timeseries [J]. IEEE Trans Vis Comput Graph 25(8):2554–2567
Lins L, Klosowski JT, Scheidegger C (2013) Nanocubes for real-time exploration of spatiotemporal datasets. IEEE Trans Vis Comput Graph 19(12):2456
Liu Z, Heer J (2014) The effects of interactive latency on exploratory visual analysis. IEEE Trans Vis Comput Graph 20(12):2122–2131
Liu Z, Jiang B, Heer J (2013) imMens: real-time visual querying of big data. Eurographics 32:421–430
Liu C, Wu C, Shao H, Yuan X (2019) Smartcube: an adaptive data management architecture for the real-time visualization of spatiotemporal datasets. IEEE Trans Vis Comput Graph 26(1):790–799. https://doi.org/10.1109/TVCG.2019.2934434
Mei H, Chen W, Wei Y, Hu Y, Zhou S, Lin B, Zhao Y, Xia J (2019) Rsatree: distribution-aware data representation of large-scale tabular datasets for flexible visual query. IEEE Trans Vis Comput Graph 26(1):1161–1171. https://doi.org/10.1109/TVCG.2019.2934800
Miranda F, Lins L, Klosowski JT, Silva CT (2017) Topkube: a rank-aware data cube for real-time exploration of spatiotemporal data. IEEE Trans Vis Comput Graph 24(3):1394–1407
Miranda F, Lage M, Doraiswamy H, Mydlarz C, Salamon J, Lockerman Y, Freire J, Silva CT (2018) Time lattice: a data structure for the interactive visual analysis of large time series. Comput Graph Forum 37(3):23–35
Moritz D, Fisher D, Ding B, Wang C (2017) Trust, but verify: optimistic visualizations of approximate queries for exploring big data. In: Proceedings of the 2017 CHI conference on human factors in computing systems, pp 2904–2915
Moritz D, Howe B, Heer J (2019) Falcon: balancing interactive latency and resolution sensitivity for scalable linked visualizations. In: Proceedings of the 2019 CHI conference on human factors in computing systems, pp 1–11
Pahins CA, Stephens SA, Scheidegger C, Comba JL (2016) Hashedcubes: simple, low memory, real-time visual exploration of big data. IEEE Trans Vis Comput Graph 23(1):671–680
Pahins CA, Ferreira N, Comba JL (2019) Real-time exploration of large spatiotemporal datasets based on order statistics. IEEE Trans Vis Comput Graph 26(11):3314–3326
Rahman S, Aliakbarpour M, Kong HK, Blais E, Karahalios K, Parameswaran A, Rubinfield R (2017) I've seen enough: incrementally improving visualizations to support rapid decision making. Proc VLDB Endow 10(11):1262–1273
Rossi RA, Ahmed NK, Zhou R, Eldardiry H (2018) Interactive visual graph mining and learning. ACM Trans Intell Syst Technol (TIST) 9(5):1–25
Shazeer N, Mirhoseini A, Maziarz K, Davis A, Le Q, Hinton G, Dean J (2017) Outrageously large neural networks: the sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538
Turkay C, Pezzotti N, Binnig C, Strobelt H, Hammer B, Keim DA, Fekete J-D, Palpanas T, Wang Y, Rusu F (2018) Progressive data science: potential and challenges. arXiv preprint arXiv:1812.08032
Vartak M, Rahman S, Madden S, Parameswaran A, Polyzotis N (2015) SEEDB: efficient data-driven visualization recommendations to support visual analytics. Proc VLDB Endow 8(13):2182–2193
Wang Z, Ferreira N, Wei Y, Bhaskar AS, Scheidegger CE (2017) Gaussian cubes: real-time modeling for visual exploration of large multidimensional datasets. IEEE Trans Vis Comput Graph 23(1):681–690
Wang Z, Cashman D, Li M, Li J, Berger M, Levine JA, Chang R, Scheidegger C (2021) Neuralcubes: deep representations for visual data exploration. In: 2021 IEEE international conference on big data (big data), pp 550–561. IEEE
Xia J, Lin W, Jiang G, Wang Y, Chen W, Schreck T (2021) Visual clustering factors in scatterplots. IEEE Comput Graph Appl 41(5):79–89. https://doi.org/10.1109/MCG.2021.3098804
Xia J, Zhang Y, Song J, Chen Y, Wang Y, Liu S (2022) Revisiting dimensionality reduction techniques for visual cluster analysis: an empirical study. IEEE Trans Vis Comput Graph 28(1):529–539. https://doi.org/10.1109/TVCG.2021.3114694
Xie C, Zhong W, Xu W, Mueller K (2018) Visual analytics of heterogeneous data using hypergraph learning. ACM Trans Intell Syst Technol (TIST) 10(1):1–26

Xu T, Zhang X, Claramunt C, Li X (2018) Tripcube: a trip-oriented vehicle trajectory data indexing structure. Comput Environ Urban Syst 67:21–28
Ying Z, Luhao G, Huixuan X, Genghuai B, Zhao Z, Qiang W, Yun L, Yuchao L, Fangfang Z (2022) ASTF: visual abstractions of time-varying patterns in radio signals. IEEE Trans Vis Comput Graph. https://doi.org/10.1109/TVCG.2022.3209469
Yuan J, Chen C, Yang W, Liu M, Xia J, Liu S (2021) A survey of visual analytics techniques for machine learning. Comput Vis Media 7(1):3–36. https://doi.org/10.1007/s41095-020-0191-7
Zgraggen E, Galakatos A, Crotty A, Fekete J-D, Kraska T (2016) How progressive visualizations affect exploratory analysis. IEEE Trans Vis Comput Graph 23(8):1977–1987
Zhao Y, Shi J, Liu J, Zhao J, Zhou F, Zhang W, Chen K, Zhao X, Zhu C, Chen W (2021a) Evaluating effects of background stories on graph perception. IEEE Trans Vis Comput Graph https://doi.org/10.1109/TVCG.2021.3107297
Zhao Y, Zhang J, Fu C-W, Xu M, Moritz D, Wang Y (2021b) Kd-box: line-segment-based kd-tree for interactive exploration of large-scale time-series data. IEEE Trans Vis Comput Graph 28(1):890–900