Iris: Amortized, Resource Efficient Visualizations of Voluminous Spatiotemporal Datasets

Kevin Bruhwiler
Kevin.Bruhwiler@rams.colostate.edu
Colorado State University
Fort Collins, Colorado

Thilina Buddhika
thilinab@cs.colostate.edu
Colorado State University
Fort Collins, Colorado

Shrideep Pallickara
Shrideep.Pallickara@colostate.edu
Colorado State University
Fort Collins, Colorado

Sangmi Lee Pallickara
Sangmi.Pallickara@colostate.edu
Colorado State University
Fort Collins, Colorado

ABSTRACT

The growth in observational data volumes over the past decade has occurred alongside a need to make sense of the phenomena that underpin them. Visualization is a key component of the data wrangling process that precedes the analyses that informs these insights. The crux of this study is interactive visualizations of spatiotemporal phenomena from voluminous datasets. Spatiotemporal visualizations of voluminous datasets introduce challenges relating to interactivity, overlaying multiple datasets and dynamic feature selection, resource capacity constraints, and scaling. In this study we describe our methodology to address these challenges. We rely on a novel mix of algorithms and systems innovations working in concert to ensure effective apportioning and amortization of workloads and enable interactivity during visualizations. In particular our research prototype, Iris, leverages sketching algorithms, effective query predicate generation and evaluation, avoids performance hotspots, harnesses coprocessors for hardware acceleration, and convolutional neural network based encoders to render visualizations while preserving responsiveness and interactivity. We also report on several empirical benchmarks that demonstrate the suitability of our methodology to preserve interactivity while utilizing resources effectively to scale.

CCS CONCEPTS

• **Information systems** → **Spatiotemporal Data**; • **Human-centered computing** → **Visualization**; • **Theory of computation** → *Sketching Algorithms*; • **Computing methodologies** → *Neural networks.*

KEYWORDS

Spatiotemporal Data; Visualization; Sketching Algorithms; Neural Networks

BDCAT '20, December 7–10, 2020, Leicester, UK

1 INTRODUCTION

The proliferation of observational devices, improvements in the resolution and frequency at which these measurements have been made, and falling costs for data storage have all contributed to an increase in data volumes. These data volumes hold the potential to unlock insights via data analytics. A key intermediate step in the data wrangling process that precedes the analyses is visualization. Visualization allows scientists to quickly assess broad patterns in the data.

This study focuses on spatiotemporal data where data is tagged with spatial information representing the location being observed and the timestamp reflecting when these measurements were made. These spatiotemporal data represent a substantial portion of the cumulative data volumes. Such data occurs in social media, observational and telemetry settings, transportation networks, simulations, and commerce among others. Spatiotemporal visualization allows us to identify spatial extents, temporal segments, or some combination thereof — also referred to as the spatiotemporal scope of interest. Identifying such spatiotemporal scopes allows practitioners to target their modeling efforts more precisely.

To maximize interoperability across different platforms we use the browser as the primary gateway for visualizations. Unlike traditional client-server interactions our visualizations entail interactions with a server-side that encompasses a distributed collection of machines. The distributed server-side is responsible for managing requests from multiple, diverse clients concurrently.

1.1 Challenges

Enabling interactive visualizations of voluminous spatiotemporal datasets involves several challenges that include:

- **Interactivity**: To be useful, visualizations must be interactive, allowing practitioners to maintain their chain-of-thought and enabling visualizations to inform their explorations. The visualizations must preserve this interactivity during key explorative operations such as drill-down, roll-up, and panning across spatiotemporal scopes.
- **Selective Overlays**: Often patterns are easier to visualize when the analyses is supplemented by auxiliary datasets. In particular, this involves allowing overlaying of features (also referred to as independent variables) from diverse datasets.

- **Data Volumes**: Visualization involves a mix of disk accesses, network transfers, computation of visual artifacts, and memory residency. Data volumes exacerbate these aforementioned challenges by exceeding resource utilization thresholds or capacities.
- **Scale**: Visualization systems must scale with increases in the number of clients. This entails minimizing interactions between the client and the server. Functionality must be effectively apportioned so that the client-side takes on a large portion of the load.

1.2 Research Questions

The broader research question that guides our investigations is the following: How can we preserve interactivity when performing spatiotemporal visualizations over voluminous datasets? Within this broader context, we have identified key research questions that we explore in this study.

- How can we effectively harness available resources and amortize workloads (CPU, memory, disk, and network I/O)? Effective amortization of workloads guides this investigation.
- How can we leverage learning during visualization? There are two aspects to this. The first aspect involves predicting spatiotemporal paths during visualization. The second aspect involves leveraging deep learning to render the phenomena, rather than computing the polygons comprising the visualization.
- How can we leverage perceptual characteristics/limits of visualization to improve the timeliness of rendering operations? This involves rendering coarser visual artifacts that are then incrementally refined.

1.3 Methodology Summary

To ensure effective visualizations our methodology addresses challenges associated with data volumes. We leverage a novel mix of algorithms and systems innovations working in concert to enable effective apportioning and amortization of workloads to preserve interactivity.

To cope with data volumes, we leverage the Synopsis sketching algorithm [1]. Synopsis is a single-pass sketching algorithm which produces statistical sketches from the data. Once the sketch is constructed it is the sketch and not the original data that is consulted during visualization operations. The sketch is a space-efficient data structure that serves as a surrogate for the voluminous data that it sketches. In particular, the Synopsis algorithm extracts information from the data and tracks distributional characteristics of the data, summary statistics, and cross-feature covariances. We have extended the basic Synopsis algorithm to facilitate efficiency gains from a compaction perspective. The sketch is broken up into a collection of space-efficient *strands*. Each strand encapsulates information for a particular spatiotemporal scope. The strands are amenable to aggregation and two strands can be combined to produce a single strand for the larger spatiotemporal scope.

Queries underpin the effectiveness of visualization operations. During visualization, Iris generates a series of queries. These higher-level queries are converted to a series of well-formulated predicates that benefit from pipelining, dispersion, and query plan optimizations on the server-side. In particular, the service query evaluation framework is designed to preserve high throughput by leveraging pipelining, reducing search space during distributed query evaluations, and ensuring frugal memory footprints. Queries are evaluated server-side and portions of the sketch that satisfy these queries are returned back to the client. Results from the queries are processed at the client side to compute visual artifacts that must be rendered.

The visualization is rendered using multiple, dynamic visualization artifacts that collectively comprise the viewport. The visualization is rendered as a Choropleth and represented as a collection of polygons that collectively depict the phenomena. Choropleths render phenomena based on natural or administrative boundaries with regions filled in using gradients determined by their feature radius. Calculation of visual artifacts is performed on a per strand basis. In particular, each strand within the sketch represents a spatial scope with a distribution of values for different feature and correlations. Each spatiotemporal scope is partitioned into a smaller set of polygons. The number of polygons is dependent on the spatiotemporal scope and the zoom level.

Polygons are used to render values for the particular spatial scope that is currently visible on the map. The rendering process consults the global gradient range that is established for a given feature. During visualizations, the Iris front-end identifies the spatiotemporal scopes that have been impacted by a visualization change. During spatiotemporal roll-up operations the polygons are aggregated, while during drill-down and panning operations the polygons are recomputed.

Iris uses a proxy-based scheme that serves as a conduit for query evaluations. Queries and responses are funneled back and forth based on gRPC and Protocol Buffers to ensure performant, compact, and low-latency communications between the client and server-side. These interactions are agnostic of the language used at either the client or server-side. For example, in our reference implementation our JavaScript based client interoperates with the Java based server-side implementation. The server-side proxy is stateless and is able to perform horizontal scaling maneuvers during large load fluctuations.

Our server-side is designed to enable high query throughputs while maintaining a lower query latency. Storage nodes are arranged as a distributed hash table (DHT) to provide better scalability and fault-tolerance. Our data dispersion criterion ensures both near-uniform distribution of data across the DHT while preserving temporal locality, which reduces disk IO during query evaluation. Data is indexed both spatially and temporally for faster lookups. Further, we leverage multi-core architectures, disk caching, and fast data serialization schemes effectively for efficient query processing.

Preserving responsiveness is a key requirement during visualizations. Traditional batched visualizations are easy to implement and reason about. However, batched visualizations have inefficiencies that stem from synchronization barriers that exist between each phase (query, retrieval, and rendering) of the visualization. Batched visualizations introduce lag because each phase cannot start before the preceding one fully completes, contributing to prolonged wait times. Furthermore, batched visualizations are computationally expensive and require all data to be available before performing

computations. As a result, they suffer resource spikes that induce failures.

In Iris, rather than retrieve results at all once, the results are streamed from the server to the client. The rendering computations are aligned with this streaming. In particular, the query retrievals and rendering operations are continually interleaved, relieving resource spikes. Furthermore, because our incremental rendering operations amortize the workloads associated with rendering operations, responsiveness is preserved as well. This process is combined with quantization (where we reduce the precision of features) when computing the visual artifacts and rendering. These are incrementally refined as additional data become available.

Leveraging deep learning for visualizations involves two key steps: encoding and mapping. We use an encoder to first learn a compact, lower-dimensional encoding of a given spatiotemporal range that can then be used to create a visualization with high fidelity. The second step involves a mapping phase where the encoded feature space is combined with a user query to produce the visualization. This approach replaces piece-wise calculation of visual artifacts within an image with a series of matrix/tensor-based operations that are highly amenable to acceleration on coprocessors at the client side. The computationally expensive operation of training the encodings is performed on the server-side away from the critical path of client interactions. Only trained models are installed on the client side. Besides reduction in memory and network overheads, another benefit of our methodology is that, because geohashes are included as a feature vector, we are able to reconcile spatial heterogeneity.

Our methodology makes effective use of resources at the client side. In particular, we leverage GPU libraries both during rendering operations and also during inferences performed by our deep CNN (convolutional neural network) based encoder. Our systems benchmarks (in section 4.1) demonstrates the suitability of our methodology to leverage hardware acceleration at the client side.

1.4 Paper Contributions

Our methodology facilitates visualization of voluminous spatiotemporal datasets at scale. In particular, our contributions include:

(1) Effective distribution of visualization workloads to reduce strain on the server side, minimize network communications, and alleviate memory pressure.
(2) We leverage co-processors (GPU) to make effective use of client-side capabilities both during calculation of visual artifacts and deep learning operations at the client side (inferences) and server-side (model training).
(3) A query processing scheme that combines evaluation over sketches with effective serialization and deserialization of sketches aligned with the visualization schemes
(4) Effective amortization of workloads by combining streaming of results with incremental refinement of the visualization to balance responsiveness. Does not introduce hotspots because the computations are amortized.
(5) A deep learning-based framework to generate visualizations on the fly. Our methodology performs a novel mapping from the feature space to the latent encoding space to produce these visualizations.

2 RELATED WORK

In previous work we looked at this problem primarily from a server-side perspective. The Aperture[2] system did not scale because visual artifacts were computed server-side. The server was also required to maintain a large and complex state to enable caching and predictive queries. This placed a lot of strain on the server and inhibited efficient horizontal scaling. In Iris we address these issues by using a simplified, stateless server and alleviating the client's computational load with data streaming, query predicate optimization, deep learning, and co-processor utilization.

Creating visualizations with streaming data has been explored in previous works with specialized techniques for time-series data[3], creating visualizations in parallel with simulations[4], and using kernel density estimation functions and GPU support[5]. Iris synthesizes several of these ideas, including leveraging the horizontal scalability of distributed computing clusters to maximize bandwidth and using client GPUs to improve rendering times. Iris also takes advantage of advances in computing technology, including efficient serialization with protocol buffers[6] and browser-GPU inter-operation with WebGL[7] and HTML canvas[8], to achieve improved performance.

Several methods of incremental visualization generation are able to provide visual analysis of datasets that are prohibitively slow or expensive to visualize in full[9–13]. However, they still suffer from one of the primary drawbacks of batched visualization: a certain amount of data must be retrieved before visualizations are recomputed and rendered. In Iris, we present an efficient method for incrementally refining visualizations as data points arrive, preserving both interactivity and high-resolution.

Much research has been done into using deep neural networks to generate images. Most notably including style transfer[14], variational autoencoders[15] (VAEs), pix2pix translation models[16], and both deep convolutional and conditional generative adversarial networks[17, 18] (DCGAN, CGAN). While these methods focus on either converting between images or sampling them from a pre-defined distribution, we replace the sampling distribution with user-defined queries and re-purpose the convolutional nature of these techniques to quickly generate geospatial visualizations.

Sketch-based storage of spatiotemporal data has been explored before [1, 19, 20]. The Synopsis sketching algorithm has been used previously to build a spatiotemporal data store. Its in-memory storage model is in contrast to the on-disk storage model developed for Iris. On-disk storage improves the scalability of the system due to the capacity differential of physical memory and disk space available in commodity hardware clusters. On the other hand, Synopsis benefits from in-memory data structures to support low latency queries. Iris relies on various optimization techniques such as parallel disk IO, parallel query execution, and disk caches to counter the IO speed differential between spinning disks and memory. Aggregate RB trees (aRB trees) [19] are used to answer spatiotemporal count queries. Iris stores spatial regions in R-trees as bounding rectangles. Each bounding rectangle points to a B-tree where historical aggregates of the feature values are stored. Tao et al. [20] extends

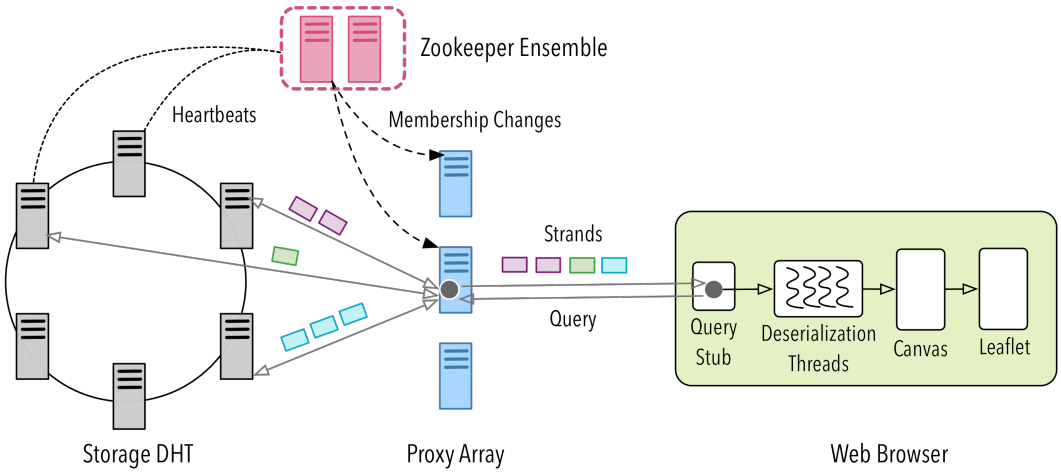

Figure 1: The high-level architecture of our query model. Proxy servers are responsible for propagating a query to multiple storage nodes, aggregating the response streams into a single stream, and sending the results back to the query client running on a browser. A small Zookeeper cluster is used to identify newly added and failed nodes.

aRB trees with the FM sketching algorithm [21] to answer distinct count queries. However, unlike Iris, these systems are designed to work with a single feature stream and support only count based queries.

NoSQL storage frameworks such as Galileo [22], MongoDB [23], and Redis [24] have been used to store spatiotemporal data. Our Sustain DHT forms the crux of our server-side data management and stores sketches of the data rather than data/documents managed by the aforementioned frameworks. Galileo and MongoDB both leverage geohashes but are focused on block and document storage respectively. Unlike Galileo, the (de)serialization schemes in the Sustain DHT are language agnostic. Redis is an in-memory storage system; our Sustain DHT manages the speed differential of the memory hierarchy via selective residency of sketches and indexes - as a result, memory footprints in the Sustain DHT are significantly smaller.

3 METHODOLOGY

Iris was developed and evaluated using data from the NOAA North American Mesoscale (NAM) forecast system [25], a large and geospatially dense set of measurements, under the assumption that few other geospatial datasets will be as difficult or computationally taxing to visualize.

3.1 Strands

The design premise of the Synopsis sketching algorithm [1] is based on representing individual feature values at a coarser-grained resolution while preserving inter-feature relationships in order to reduce data size. This process is called *discretization*. Discretization represents each feature value as a record using a bin based on a predetermined bin configuration. A bin configuration comprises a set of non-overlapping intervals that collectively construct the range for a particular feature.

For instance, suppose the bin configuration for surface temperature (in Kelvin) is `[217.9, 256.7)`, `[256.7, 285.8)`, `[285.8, 305.2)`, `[305.2, 334.3)`. If the surface temperature is `220.31` of a particular record, then it falls within the range of the first bin, `[217.9, 256.7)`, therefore represented using the first bin. Similarly,

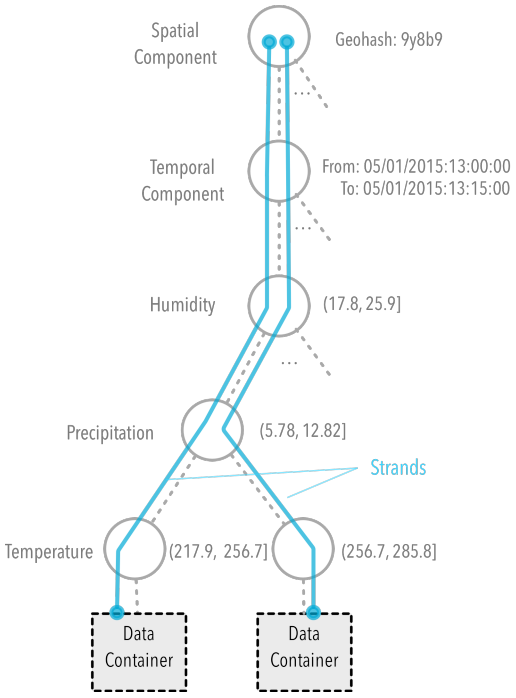

Figure 2: An example of strand construction using Synopsis sketching algorithm. Spatio-temporal components and individual features are represented in coarser-grained resolutions to reduce the storage footprints. In this example, the subtree rooted at the vertex for temporal component summarizes all the records occurring in an area of $4.9 \times 4.9 km^2$ represented by the geohash `9y8b9` for the 15 minute time interval starting at `13:00:00` on `05/01/2015`.

if the temperature slightly increases to `220.32` in the next record, the updated feature value still gets mapped to the same bin.

In real settings, bin configurations are more dense — most bin configurations contain around 30-50 bins. Bin configurations are calculated using a kernel density estimation based method such that the overall error due to discretization is maintained below a given threshold. We use the normalized root mean square error (NRMSE) of 2.5% as our error threshold for the benchmarks. Further, our storage framework supports the evolution of bin configurations over time to capture concept drifts (e.g., different bin configurations for Winter and Summer seasons for a given region).

Spatial and temporal components of observation are also represented at coarser resolutions using geohashes. The geohash algorithm [26] produces a deterministic mapping of 2-dimensional spatial extents into a 1-dimensional string; the length of the geohash string is inversely proportional to the size of the spatial extent – the longer the geohash, the smaller the spatial extent that it represents. We use a prefix of the geohash to represent the spatial component — this maps multiple observations that occur within spatial proximity onto the same geohash prefix. Similarly, temporal components are also mapped to coarser-grained time intervals.

The discretized records are then collated into a tree-like data structure, as shown in Figure 2. Each tree path ending with a unique leaf node is called a *strand*. Coarser-grained resolutions improve the compaction of the dataset by mapping multiple records into fewer strands. Records that occur within a particular spatial and temporal proximity will be likely to be mapped into a single strand after being discretized. At leaf nodes, a set of online statistics are maintained to summarize all records that are represented using that particular strand. We use Welford's method [27] to maintain the number of observations, the running mean, the sum of squares of differences from the current mean, and the sum of cross products between features. These statistics are useful for calculating descriptive statistics as well as inter-feature relationships. We use strands as the unit of data storage in our backend system.

3.2 Query Generation and Refinement

3.2.1 Spatial Predicates. The spatial predicates for each query are determined by a geohash-based tree search. The smallest bounding geohash is computed by taking the longest matching geohash of the top-right and bottom-left corners of the visualization area. For example, if the top-right geohash is "9zgvkpbsf" and the bottom-left is "9zjs22e3h", the shortest matching prefix would be "9z". Sub-geohashes are then searched recursively: the bounding box of each geohash is compared to the bounding box of the visualization area to determine if there is any overlap. If the boxes do overlap, that geohash is added to the search set and the process continues down to a predefined precision. Due to overheads in querying many geohashes simultaneously the target precision is determined based on the current zoom level of the visualization so that there will never be too many geohashes in a single query.

3.2.2 Temporal Predicates. The temporal range of a query is determined based on two factors: the frequency of the sketched data and a user-selected time. All sketched data is bucketed into a configurable temporal bracket (see Section 3.3). Each query specifies a single temporal bracket, although querying multiple brackets is possible. The bracket is determined via a Time-Dimension UI[28] that allows users to query a specific time or play an animation across a range of times by performing many sequential queries.

3.2.3 Speculative Query Generation. Iris implements a speculative query methodology similar to Aperture's[2] where some visualizations are pre-rendered. Speculative queries are performed on both a temporal basis, in which future times are queried, and a spatial basis, in which the user's panning trajectory is linearly extrapolated. Temporal extrapolation is done by simply incrementing the query time by the temporal bracket, while spatial extrapolation is done by examining the previous two visualization areas and querying the next area in a straight line. However, unlike Aperture, Iris cannot rely on the server to generate concurrent visualizations, and simultaneously computing multiple visualizations reduces client responsiveness (profiled in section 4.1). Consequently, speculative queries are only performed while no other visualization is being actively generated. Precomputed visualizations are stored in an LRU cache on the client and rendered as needed.

3.3 Query Evaluation

Our storage subsystem is optimized for queries (read traffic). We employ an array of optimizations such as uniform data dispersion, data locality, indexing, parallel query execution, parallel disk I/O, and caching for efficient data retrieval. Figure 1 depicts the high level architecture of our query model.

We arrange the storage nodes as a DHT with a consistent hashing-based data dispersion scheme based on a key generated by combining the spatial attribute and the temporal attribute of a strand. DHTs provide better load balancing, incremental scalability, ability to work with heterogeneous commodity hardware, and fault tolerance. We use the geohash of a strand combined with a coarser-grained temporal component (e.g., the month of the year) to generate the key for data dispersion. Using a coarser-grained temporal component, instead of the higher resolution timestamps, provides better temporal locality during data storage. Consistent hashing on a key generated by combining both spatial and temporal components and the use of virtual nodes [29] enables better load balancing across the DHT.

At each DHT node, strands are indexed both spatially and temporally for efficient retrieval. Geohashes are indexed using a prefix tree to support efficient wildcard matching. Leaf nodes of the prefix tree are log structured merge (LSM) trees where strands are stored in a sorted order based on their timestamps. Each LSM tree contains a hierarchical temporal index to retrieve matching strands based on temporal predicates of a query. We utilize multiple disks available on a machine for storage to facilitate parallel disk I/O.

During query evaluation, a query is transformed into multiple sub-queries by partitioning the spatial scope. These sub-queries are then evaluated in parallel to leverage the multi-core architecture in the underlying hardware. We also leverage disk caches by pinning frequently accessed data in memory, a technique used in blob-store implementations [30]. Memory limits are enforced on the storage node processes such that the operating system can use sufficient physical memory for the disk cache. Parallel query execution works with caching and parallel disk I/O to reduce the query latencies significantly.

3.4 Streaming Results

The cumulative network footprint of the strands matched with a particular query can vary from a few Kilobytes to hundreds of Gigabytes depending on the spatiotemporal scope collectively constructed by the predicates. Instead of returning all matching strands at once, we stream matching strands to the client as soon as they are retrieved by the worker threads.

Streaming results and handling them incrementally is beneficial for visualizations because: (1) the visualization is updated immediately after any user interaction, dramatically improving the perceived responsiveness of the system, and (2) expensive computations required to render the visualization are amortized, minimizing interference with other processes on the client machine and reducing the odds of the client suffering load-related failures. Further, a streaming query model reduces the strain on the memory of the server by shortening the memory retention period of the results, therefore improving the overall query throughput of the system.

We designed our query API to be language agnostic by using gRPC/Protocol Buffers — allowing clients implemented using numerous supported programming languages to interact with our storage system. We group multiple strands into a single message to improve the network bandwidth utilization.

The query API is exposed through an array of proxy services acting as the gateway to the storage system. Once a query is received, the proxy server forwards it to the DHT, triggering a set of response streams originating from individual DHT nodes. These streams are then merged into a single stream at the proxy server and funneled back to the client. Proxy nodes can often become bottlenecks due to saturated resources such as network bandwidth and CPU. To counter this issue, new proxies can be added on-demand without disrupting the ongoing queries. The stateless runtime design of proxy servers enables the seamless horizontal scaling of the proxy array.

Data arrives at the client as a stream of Synopsis strands, serialized as protocol buffers. The client is responsible for deserializing them, converting the geohash location to a *(latitude, longitude)* point, calculating the color of the associated polygon from the deserialized features, and passing this information to the HTML canvas. The client performs these operations on each data strand asynchronously to allow it to cope dynamically with changes in the rate of data arrival.

3.5 Rendering

Sketches are rendered on an HTML canvas in real-time as strands arrive at the client, allowing users to visualize data points the moment that they are available. The canvas element is a low level model in HTML used for rendering bitmaps and 2D graphics. It can optionally make use of WebGL which enables 3D rendering and hardware acceleration via a GPU.

Due to the high rate of strand arrival it is critical that the canvas can render each strand quickly. Because canvas elements can be updated incrementally, where only the modified region of the canvas is recomputed rather than the entirety, it is well-suited to rendering streamed data. Additionally, we ensure that the canvas is not performing expensive anti-aliasing operations by rounding the location of each polygon so that its bounds align evenly with the canvas' pixels.

On the canvas, each strand is represented as a square polygon of a dynamic size, the color of which is determined by the strand's features. The size of each polygon is computed using one of several decay algorithms (see Section 4.2): the first few polygons are extremely large, taking up much of the visualization area, while the size of later polygons is rapidly reduced down to a fixed minimum. This has the effect of immediately providing users with a coarse-grained visualization that becomes increasingly refined as more data arrives. The incremental rendering process is visualized in Fig. 4.

3.6 Deep Neural Networks

We experiment with using deep convolutional networks to quickly generate visualizations on the client. We explore the ways in which this can improve the performance of Iris in three major respects:

(1) Reducing network traffic by eliminating the need to re-run queries server-side when users change the spatial, temporal, or feature predicates of the visualization within a predefined temporal range.
(2) Improving the responsiveness of data visualizations by eliminating the need to wait for data to be streamed from the server.
(3) Reducing computational load at the client by replacing CPU-intensive deserialization operations with matrix multiplications highly amenable to co-processor acceleration.

We also investigate the uncertainty that comes from using generated visualizations and the potentially deleterious effects on the final resolution.

3.6.1 Training. The training dataset was created and curated by generating high quality visualizations for a large number of user queries, then saving the query and the visualization along with all of the data strands for that particular temporal scope. This resulted in a set of *(input, target)* pairs where the input consisted of both a user query and all of the strands from the encompassing temporal scope that could potentially be used to generate the associated visualization. The strands are grouped into a matrix that preserves the geospatial relations between the data points, allowing the use of convolutional layers.

The network was trained in a distributed computing cluster using PyTorch[31], both to make use of additional cores and help cope with the relatively slow process of reading and evaluating the (input,target) pairs. The error was calculated by averaging the binary cross-entropy loss[32] (BCE) between every pixel in the generated image and the target visualization. Gradients were computed to minimize the BCE loss and the weights of the network were updated using the Adam[33] stochastic optimizer.

3.6.2 Network Structure. The network is structured as a typical image-to-image translation model, using convolutional layers to transform one image to another. The user query is encoded as an additional channel and appended to the matrix of input strands.

Due to the limited types of matrix operations that can be executed on the client (see section 3.6.3) the generator is required to have an unusual structure. Rather than using the traditional up-sampling or

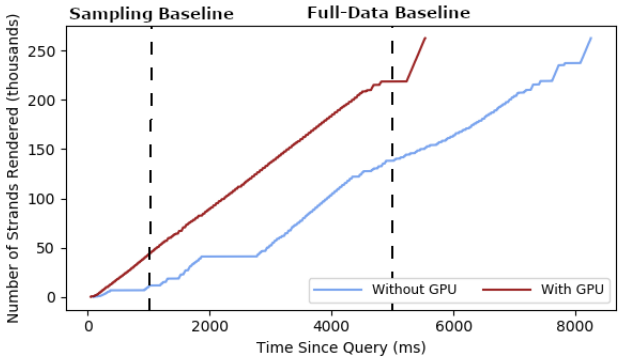

Figure 3: A comparison between query times with and without GPU support. Using a GPU, Iris is able to render data at a faster and more consistent rate.

deconvolutional layers to generate images common in GANs and VAEs, we use a series of convolutional layers with a low number of filters that do not change the size of the image. This allows us to transform data into a generated rendering within the contraints of web-based deep learning while still taking advantage of the spatial nature of convolutions.

3.6.3 Mapping Phenomenon. To run the trained network on the client-side we first convert the trained PyTorch model to Onnx[34], a protocol buffer based machine learning model representation format. We use Onnxjs, which has WebGL support, to execute the model on the browser with co-processor support. Onnxjs was chosen over Tensorflowjs[35] due to higher compatibility with different libraries and improved speed during model evaluation[36].

4 EVALUATION

Experimental Setup: The evaluation was performed on a client with 32GB of RAM, an Intel Core i5-9300H 2.40GHz CPU, and a Nvidia 1660GTX GPU. Iris was accessed via a Chrome browser, and GPU support was disabled during some experiments by disabling WebGL canvas acceleration.

The data storage system comprised 75 nodes (Xeon E5-2620, 32 GB Memory, 4 TB storage) and a varying number of proxy servers (Xeon E5-2620, 64 GB Memory) depending on the experiment, each running Fedora 30 and JDK 1.8.0_251. Each machine is connected to the cluster using a 1 Gbps link.

4.1 Responsiveness

Responsiveness in Iris is difficult to quantify due to the incremental nature of the visualization construction. It is also difficult to compare to other systems as there are few specialized geospatial data visualization tools, none incremental. GeoSparkViz[37] is capable of rendering geospatial scatter-plots and heat-maps at a rate of 20 million points per minute, moderately faster than Iris, however doing so requires the visualization to be generated server-side using all the cores in a Spark cluster, restricting it to one user at a time, and suffers from the overheads of submitting jobs and collecting results, which make sub-second latencies impossible. Interactive geospatial visualization systems, such as Aperture or Waldo[38], are capable of rendering continent-scale visualizations in 5-7 seconds. Some visualization systems, such as ISOS[39], achieve real-time interactivity with sampling algorithms. As a heuristic for comparing Iris to the different types of geospatial visualization tools, we define two baselines: the sampling baseline, at one second, and the full-data baseline, at five seconds.

As can be observed in Fig. 3, the first data points arrive a fraction of a second after a query is issued. Although Fig. 4 shows that a visualization is not completed immediately, the rapid feedback is likely to improve user perception of the responsiveness of the system[40], as is the visible and constant improvement as the visualization is refined[41]. We can also see that, with co-processor support, Iris is able to match the full-data baseline.

Fig. 3 also quantifies the difference between running Iris with and without a GPU. The presence of a co-processor dramatically improves the performance of Iris, reducing the time to complete a visualization by more than 30%. Additionally, it stabilizes the rate at which the visualization is incrementally rendered, likely because the CPU faces contention from threads responsible for receiving and deserializing data strands.

Finally, Iris implements a similar caching scheme to Aperture in which speculative queries are made to pre-load data that a user is likely to view in the future. However, Aperture did so by relying on the server to both concurrently render and cache speculative queries, while Iris' stateless server and client-side rendering makes that impossible. Experimentation shows that rendering multiple visualizations simultaneously on the client (using an offscreen canvas[8]) has a significant detrimental impact on client responsiveness, with each additional rendering causing a linear decrease in performance (two simultaneous renderings take twice as long, three take thrice as long, etc...) regardless of the number of streams. As a result Iris only runs speculative queries one at a time, and only when there is no rendering being actively performed. But, unlike Aperture, Iris is capable of rendering high-quality visualizations with sub-second latency without the use of caching, making speculative queries non-critical.

Processor	Mean Evaluation Time (ms)	Standard Deviation
CPU	2096.75	64.67
GPU	231.60	135.04

Table 1: Comparison in neural network evaluation times with and without GPU support, averaged over 20 runs.

4.1.1 Responsiveness with Learned Visualizations. We also evaluate the responsiveness of queries rendered using the pre-trained deep neural network. The results of that evaluation are shown in Table 1. The presence of a GPU improves performance by nearly a factor of ten, and, similarly to the incremental rendering operations, performing the evaluation on a GPU reduces to competition between simultaneous rendering and deserialization operations.

4.2 Fidelity

Iris is able to generate visualizations with sub-second latency due to a decaying-size rendering scheme in which early data strands are used to approximate the values of spatially adjacent data strands until those strands arrive at the client. This results in a loss in

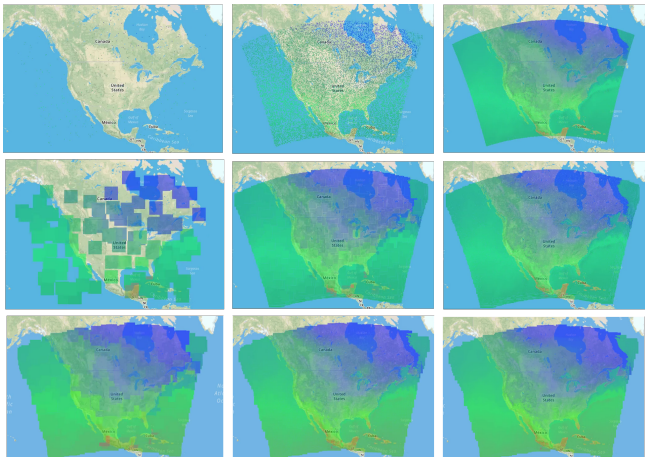

Figure 4: Three examples of increasing resolution over time. The top row is without decay, the middle row uses linear decay, and the bottom row uses exponential decay. Visualizations are shown at $1/10^{th}$ of a second after the query, $3/4^{ths}$ of a second after the query, and 4 seconds after the query.

visualization fidelity, as illustrated in Fig. 5. Both exponential and linear decay schemes are never able to reach optimal fidelity due to approximations made with early data strands. In Fig. 4 it can be seen that the loss in fidelity occurs around the edges of the dataset where data is scarce. In these regions certain strands are made to approximate data that does not exist, creating some areas of the visualization that will never be refined.

The response time gain, however, is enormous, with the exponential decay scheme reaching its maximum fidelity in a quarter of a second, easily under both the sampling and full-data baselines. Assuming that the area being visualized is not too close to the spatial edge of the dataset, the small loss in fidelity is visually insignificant (see Fig. 4) compared to the improvement in response time.

4.2.1 Fidelity of Learned Visualizations. Fig. 5 also shows the quality of the pre-trained visualizations that are rendered as more data

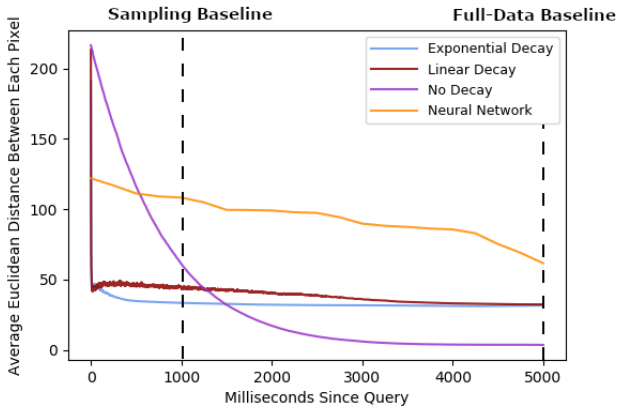

Figure 5: A measurement of how image fidelity improves over time with different methods. Comparisons are between the current visualization and the final visualization generated without decay.

arrives. All the strands that have arrived so far are evaluated and a learned visualization is generated every half-second, as the model evaluation is too slow to be done every time a new strand arrives. In Fig. 5, the generated raster shows moderately higher error than any of the decay schemes, although it does outperform manual rastering when just a few strands have arrived due to the fact that, even with very little data, the model "knows" where data points are located and the relative temperatures of varying regions.

4.3 Supporting low latency queries

We have designed our data storage system and the associated query processing system to facilitate low latency and scalable query evaluation. We validate our design decisions outlined in Section 3.3 using a set of micro-benchmarks and system benchmarks.

Given that strands are disk-bound data, we leverage an array of disks attached to a node to provide higher disk I/O (both reads and writes). Figure 6 depicts the cumulative disk read throughput achievable with multiple disks attached to a single node. Near-linear increase in the cumulative read throughput implies the minimal overhead at the data access and query processing layer. In our experimental setup, the network bandwidth is the most constrained resource, which indirectly limits the disk read throughput.

Figure 7 and Figure 8 demonstrate the effect of parallelized query processing and disk caching. At every DHT node, a query is split into a series of subqueries based on the qualifying spatial scopes, which are then processed in parallel. This approach is most effective with larger geospatial scopes — query completion times are improved by 74.7% for geohash prefixes of length 0, whereas the improvement is 38.9% for geohash prefixes with length 4. On the other hand, we observed a consistent improvement, 96.4% - 98.3%, due to disk cache irrespective of the geohash prefix length, as shown in Figure 8.

We profiled the query performance of our storage subsystem — the results are depicted in Figure 9. Query completion times (latencies) and the cumulative query throughput of the system is recorded for a different number of concurrent queries. Each query retrieved one month's worth of data for an area represented by a geohash prefix of length 4, therefore more uniformly distributing the queries throughout the DHT. Our observations align with the behavior of a typical distributed system approaching the peak performance. Latency stays constant initially before increasing with the cumulative query throughput. Once the query throughput reaches the

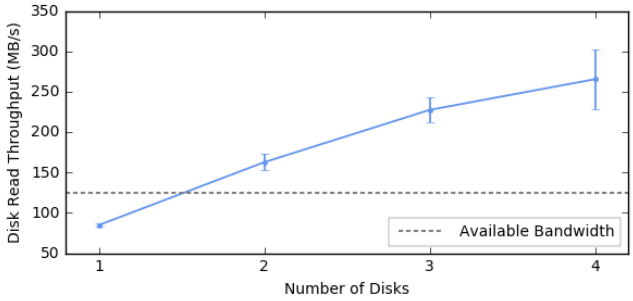

Figure 6: Disk read rates at a single node with multiple disks. Near linear growth suggests a low overhead from the data access layer.

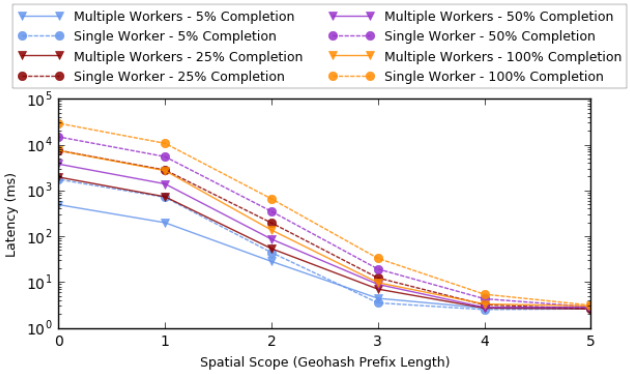

Figure 7: Impact of processing a query in parallel using multiple workers. Parallel processing significantly reduces the latency of queries with predicates corresponding to larger scopes.

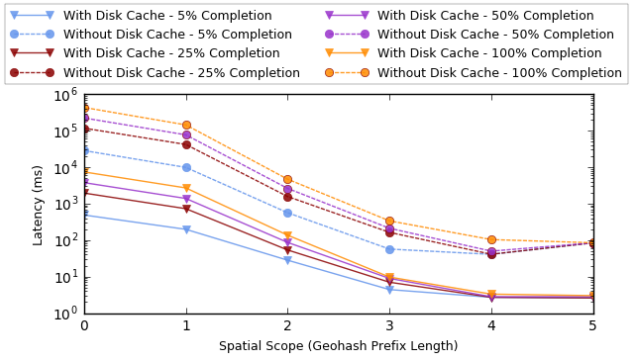

Figure 8: Our query processing model extensively relies on disk cache to reduce disk I/O. Using of caching reduces the query latency irrespective of the query scope.

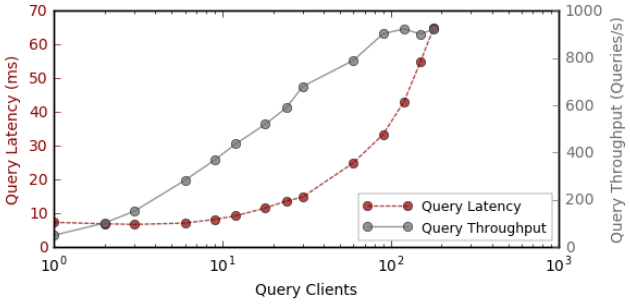

Figure 9: Query latency vs. throughput as we funnel multiple queries simultaneously through a single proxy node. Once the proxy nodes reach their maximum capacity, latency grows exponentially.

maximum capacity, the latency increases exponentially. Because the data storage system is the only aspect of Iris shared between users, all rendering is pushed to the client, this demonstrates that our experimental setup can handle as many as 100 simultaneous queries before reaching capacity. Because users are unlikely to be

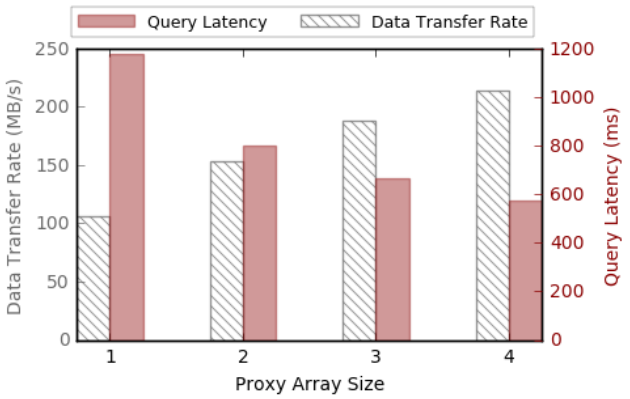

Figure 10: Demonstrating the horizontal scalability of the proxy server arrays. Incrementally adding proxy servers relieves the bandwidth bottleneck, improving latency and throughput.

making queries every half-second, this translates to thousands or tens of thousands of simultaneous clients.

Proxies act as the interface between the query clients and the backend DHT, therefore they tend to become a bottleneck under heavy load. More proxy servers are provisioned on-demand to counter this issue. This behavior is demonstrated in Figure 10 — we saturate the network bandwidth of a single proxy server using a set of query clients operating simultaneously. As more proxy servers join the array, the cumulative query output (measured in terms of data transfer rate) is increased, while latencies are improved significantly. We do not see a linear improvement of performance because we keep the initial load unchanged, which is not sufficient to saturate the network bandwidths of multiple proxy servers.

5 CONCLUSIONS AND FUTURE WORK

In this study we presented our methodology involving a mix of workload amortizations, forecasts and learning, and exploiting perceptual characteristics to render visual artifacts. In particular, Iris facilitates interactivity at the client while allowing high-throughput interactions at the server side.

RQ-1: Apportioning workloads across CPUs, GPUs, memory, network, and disk I/O enables concurrent utilization of resources. The framework leverages CPUs to perform keys aspects relating to query evaluations such as search space reduction, traversal of indexes, thread pool management and utilization of cores, and DHT operations. Leveraging GPUs allow us to accelerate rendering of visualizations and tensor operations triggered during model training and prediction. Our framework manages memory residency of indexes and reduced disk I/O during retrieval operations. Frugal use of network I/O via the use of Protocol Buffers for (de)serialization of queries results in reduced footprints for network bandwidth.

RQ-2: Leveraging access patterns enables identification of visual artifacts that can be precomputed. We also leverage patterns in the data to train models that render visualizations. Training of our deep learning models occur server-side away from the critical path of renderings; only trained models reside, and inferences occur, at the client side. Three characteristics ensure responsiveness; model inferences (1) are underpinned by operations on tensors that are

highly amenable to GPU accelerations, (2) minimize client interactions, and consequently network I/O, with the server-side, and (3) since no serialization-related operations need to be performed, there are no synchronization barriers between the CPU and GPU allowing interactive renderings on the client-side.

RQ-3: Our rendering scheme includes a cooperative mechanism involving the client and server side to combine fast coarser-grained rendering with incremental refinements that improves the visualization over time. This is underpinned by our server-side DHT framework that partitions a query into multiple predicates that are not just evaluated concurrently, but also support streaming of results as they become available. Our methodology manages and ensures a high degree of concurrency both in a distributed setting and at a particular node. The degree of concurrency at a node can be dynamically tuned based on the ongoing system load. Besides reducing interference across queries to improve responsiveness, this has the added benefit of high throughput evaluations.

As part of future work, we plan to experiment with generative adversarial networks to explore its implications on the quality and speed of rendering operations.

ACKNOWLEDGMENTS

This work was supported by grants from the U.S. National Science Foundation [OAC-1931363, ACI-1553685], the Advanced Research Projects Agency-Energy, and a Cochran Family Professorship.

REFERENCES

[1] Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2552–2566, 2017.

[2] Kevin Bruhwiler and Shrideep Pallickara. Aperture: Fast visualizations over spatiotemporal datasets. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, UCC'19, page 31–40, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Daniel A. Keim, Jörn Schneidewind, and Mike Sips. Circleview: A new approach for visualizing time-related multidimensional data sets. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '04, page 179–182, New York, NY, USA, 2004. Association for Computing Machinery.

[4] J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. C. Law, and M. Papka. Large-scale data visualization using parallel data streaming. *IEEE Computer Graphics and Applications*, 21(4):34–41, 2001.

[5] O. Daae Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *2011 IEEE Pacific Visualization Symposium*, pages 171–178, 2011.

[6] K. Maeda. Performance evaluation of object serialization libraries in xml, json and binary formats. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*, pages 177–182, 2012.

[7] Tony Parisi. *WebGL: Up and Running*. O'Reilly Media, Inc., 1st edition, 2012.

[8] Steve Fulton and Jeff Fulton. *HTML5 canvas: native interactivity and animation for the web*. " O'Reilly Media, Inc.", 2013.

[9] Danyel Fisher, Igor Popov, Steven Drucker, and m.c. schraefel. 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*, CHI '12, page 1673–1682, New York, NY, USA, 2012. Association for Computing Machinery.

[10] J. Im, F. G. Villegas, and M. J. McGuffin. Visreduce: Fast and responsive incremental information visualization of large datasets. In *2013 IEEE International Conference on Big Data*, pages 25–32, 2013.

[11] Mike Barnett et al. Stat! an interactive analytics environment for big data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, page 1013–1016, New York, NY, USA, 2013. Association for Computing Machinery.

[12] Héléna A. Gaspar, Igor I. Baskin, Gilles Marcou, Dragos Horvath, and Alexandre Varnek. Chemical data visualization and analysis with incremental generative topographic mapping: Big data challenge. *Journal of Chemical Information and Modeling*, 55(1):84–94, 2015. PMID: 25423612.

[13] Michael Glueck, Azam Khan, and Daniel J. Wigdor. Dive in! enabling progressive loading for real-time navigation of data visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 561–570, New York, NY, USA, 2014. Association for Computing Machinery.

[14] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[15] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

[16] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[17] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[18] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[19] Dimitris Papadias, Yufei Tao, P Kanis, and Jun Zhang. Indexing spatio-temporal data warehouses. In *Proceedings 18th International Conference on Data Engineering*, pages 166–175. IEEE, 2002.

[20] Yufei Tao, G. Kollios, J. Considine, F. Li, and Dimitris Papadias. Spatio-temporal aggregation using sketches. In *Proc. of the Intl. Conference on Data Engineering*, pages 214–225, March 2004.

[21] Philippe Flajolet and G Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.

[22] M. Malensek, S. L. Pallickara, and S. Pallickara. Galileo: A framework for distributed storage of high-throughput data streams. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 17–24, 2011.

[23] Kristina Chodorow and Michael Dirolf. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2010.

[24] Josiah L. Carlson. *Redis in Action*. Manning Publications Co., USA, 2013.

[25] National Oceanic and Atmospheric Administration. The North American Mesoscale Forecast System, 2020.

[26] Gustavo Niemeyer. Geohash, 2008.

[27] BP Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[28] Balearic Island Coastal Observing and Forecasting System. Leaflet.TimeDimension. https://github.com/socib/Leaflet.TimeDimension, April 2020.

[29] Giuseppe DeCandia et al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007.

[30] Shadi A. Noghabi et al. Ambry: Linkedin's scalable geo-distributed object store. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 253–265, New York, NY, USA, 2016. Association for Computing Machinery.

[31] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. pages 8024–8035. Curran Associates, Inc., 2019.

[32] Andreas Buja, Werner Stuetzle, and Yi Shen. Loss functions for binary class probability estimation and classification: Structure and applications. *Working draft, November*, 3, 2005.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[34] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019.

[35] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[36] Yulong Wang and Hariharan Sheshadre. Onnxjs. https://github.com/microsoft/onnxjs, April 2020.

[37] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. Geosparkviz: a scalable geospatial data visualization framework in the apache spark ecosystem. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2018.

[38] D. A. Keim, C. Panse, M. Sips, and S. C. North. Visual data mining in large geospatial point sets. *IEEE Computer Graphics and Applications*, 24(5):36–44, 2004.

[39] Tao Guo, Kaiyu Feng, Gao Cong, and Zhifeng Bao. Efficient selection of geospatial data on maps for interactive and visualized exploration. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 567–582, New York, NY, USA, 2018. Association for Computing Machinery.

[40] John A Hoxmeier and Chris Dicesare. System response time and user satisfaction: An experimental study of browser-based applications. *Proceedings of the Association of Information Systems Americas Conference*, 01 2000.

[41] Nicola Cranley, Philip Perry, and Liam Murphy. User perception of adapting video quality. *International Journal of Human-Computer Studies*, 64(8):637–647, 2006.