DISSERTATION

TOWARDS INTERACTIVE ANALYTICS OVER VOLUMINOUS SPATIOTEMPORAL DATA USING A DISTRIBUTED, IN-MEMORY FRAMEWORK

Submitted by

Saptashwa Mitra

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2023

Doctoral Committee:

Advisor: Sangmi Lee Pallickara

Shrideep Pallickara
Francisco Ortega
Kaigang Li

ABSTRACT

TOWARDS INTERACTIVE ANALYTICS OVER VOLUMINOUS SPATIOTEMPORAL DATA USING A DISTRIBUTED, IN-MEMORY FRAMEWORK

The proliferation of heterogeneous data sources, driven by advancements in sensor networks, simulations, and observational devices, has reached unprecedented levels. This surge in data generation and the demand for proper storage has been met with extensive research and development in distributed storage systems, facilitating the scalable housing of these voluminous datasets while enabling analytical processes. Nonetheless, the extraction of meaningful insights from these datasets, especially in the context of low-latency/ interactive analytics, poses a formidable challenge. This arises from the persistent gap between the processing capacity of distributed systems and their ever-expanding storage capabilities. Moreover, the interactive querying of these datasets is hindered by disk I/O, redundant network communications, recurrent hotspots, transient surges of user interest over limited geospatial regions, particularly in systems that concurrently serve multiple users. In environments where interactive querying is paramount, such as visualization systems, addressing these challenges becomes imperative.

This dissertation delves into the intricacies of enabling interactive analytics over large-scale spatiotemporal datasets. My research efforts are centered around the conceptualization and implementation of a scalable storage, indexing, and caching framework tailored specifically for spatiotemporal data access. The research aims to create frameworks to facilitate fast query analytics over diverse data-types ranging from point, vector, and raster datasets. The frameworks implemented are characterized by its lightweight nature, residence primarily in memory, and their capacity to support model-driven extraction of insights from raw data or dynamic reconstruction of compressed/ partial in-memory data fragments with an acceptable level of accuracy. This approach effectively helps reduce the memory footprint of cached data objects and also mitigates the need

for frequent client-server communications. Furthermore, we investigate the potential of leveraging various transfer learning techniques to improve the turn-around times of our memory-resident deep learning models, given the voluminous nature of our datasets, while maintaining good overall accuracy over its entire spatiotemporal domain. Additionally, our research explores the extraction of insights from high-dimensional datasets, such as satellite imagery, within this framework.

The dissertation is also accompanied by empirical evaluations of our frameworks as well as the future directions and anticipated contributions in the domain of interactive analytics over large-scale spatiotemporal datasets, acknowledging the evolving landscape of data analytics where analytics frameworks increasingly rely on compute-intensive machine learning models.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my advisor, Dr. Sangmi Lee Pallickara, for her unwavering support, guidance, and mentorship throughout my doctoral journey. Her expertise, encouragement, dedication, and willingness to share her knowledge have been instrumental in shaping my research and academic growth. I am profoundly grateful for her belief in my potential, and for supporting me even during times of personal and professional adversity. Her faith in my abilities have been a constant source of motivation that pushed me to excel in my research.

I would also like to extend my thanks to the entire Computer Science department staff for their invaluable assistance, resources, and a collaborative atmosphere that fostered my intellectual development. The encouragement and camaraderie I received from my fellow students and colleagues in the department have been instrumental in my academic and research endeavors.

On a personal note, I want to acknowledge the natural beauty of Colorado and the vibrant city of Fort Collins. The breathtaking landscapes, the pristine lakes, and the serene surroundings of this region have provided the perfect respite during the most challenging times of my academic journey. Colorado's natural beauty has been the best stress reliever in the world, offering moments of tranquility and inspiration, as well as ample opportunities for outdoor activities.

This journey would not have been possible without the unwavering support of parent, my family, and my friends back home in India. Their love, understanding, patience, and encouragement have been my pillars of strength throughout this endeavor. In closing, I extend my deepest appreciation to everyone who has played a part in my academic and personal growth. Your support and belief in me have been invaluable, and I am truly grateful for the privilege of completing this PhD journey with your encouragement.

This research was funded in part by the National Science Foundation (OAC-1931363, ACI-1553685, CNS-2312319), the National Institute of Food and Agriculture [COL0-FACT-2019], and an NSF/NIFA AI Institutes Award [2023-03616].

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Valuable insight can be gained from data through well-designed analytical techniques – this process is called *sensemaking*. The more expansive the underlying data or the larger the sample size represented by the data, the more potential it has in drawing inferences capable of driving decision-making, extracting patterns, or modeling phenomena [1]. This assumption underpins the current interest in research over analytics over large-scale data.

Visualization is a process that informs human perception with data analytics by mapping attributes of the raw data to human-interpretable formats, such as figures, heatmaps, charts, etc. [2] Visualization allows scientists to explore the data at *"rates resonant with the pace of human thought"* [3, 4]. Visualizations can help convey complex relationships among the data attributes in a more intuitive format, enable identification of interesting patterns, infer correlations, and causalities – this makes them a powerful tool in sensemaking. Interactive visualization entails a gradual exploration of the domain of the underlying data in sequence of exchanges between the user and the back-end server [5]. This iterative dialogue between users' and server is crucial in enabling actionable observations or pattern-detection regarding the phenomena being investigated. However, in a large-scale data-analytics setting, this back-and-forth communication between a client and server that is required to support such interactive visualization becomes a challenge due to increased latency of the server response.

The current ever-increasing scale and availability of scientific data can be attributed to an increase in the number of informational and communicational technologies (ICT), like sensors, RFIDs, meters, GPSs and other observational instruments, which has been termed as *data explosion* [6]. Majority of these data are unstructured in the form of logs, image, video, audio, and so on. These datasets are too large and complex to be stored or processed using conventional databases and analytics tools. At the same time, the size and span of these data resources provides ample op-

portunities for various scientific explorations such as machine learning, hypothecation, correlation analysis, etc., that help influence public policies and business strategies, among others.

Traditional analytics techniques that are effective over small datasets are infeasible in case of modern big data. Data-centric decision-making over large-scale data poses computational challenges given their volume, high dimensionality, and heterogeneity. Interactive analytics over them often involves high-throughput data retrievals that entail frequent and intensive network communication and disk I/O over backend servers. Such operations are constrained by limitations in available memory, computational capacity and network/disk I/O in both the servers and the clients. All these factors combined make the problem of interactive analytics over large-scale data a challenging one. Although there have been a multitude of systems capable of efficiently storing these datasets over a distributed cluster of nodes and providing query-based analytics over the distributed cluster in a fault-tolerant manner, interactivity of these analytical operations continues to be a problem once the size of the dataset involved crosses a certain threshold, such as peta-scale.

A major portion of large-scale data generated nowadays consists of spatiotemporal data, i.e., data collected across both space and time. McKinsey Global Institute says that the pool of personal location data was in the level of 1 PB in 2009 and is growing at a rate of 20% per year [7], indicating the rate of generation of these kinds of datasets over time. In this study, we focus on the computational capabilities that enable real-time visual analytics over large-scale *spatiotemporal data*. Spatiotemporal datasets are generated through continuous monitoring of phenomenon using various observational equipment. These datasets span a wide range of domains like atmospheric science, earth science, public health, ecology, epidemiology, climatology, etc. Spatiotemporal data can be in the form of raster data (eg., satellite imagery), vector data or graphs (eg., GPS data). Hence our framework needs to be versatile enough to handle such heterogeneous data.

1.1 Research Challenges

Interactive visual analysis relies on human visual and cognitive system to intuitively detect patterns or details that could otherwise go unnoticed. The interactive nature of the analysis can often

reveal more information than complex, but static visualizations [5]. **Interactivity** has a significant impact on the effectiveness of visualization for exploratory analytics since it has an impact on the attention span of end-users [8]. The primary goal of this research is to facilitate *effective and rapid query evaluation on large spatiotemporal datasets while ensuring low latencies, fidelity, and representativeness*. Our research aims to enable the end-users to explore the large-scale spatiotemporal data without being constrained by the volume and complexity of the raw observations stored in the back-end. The following are the challenges associated with such datasets.

1.1.1 Visual Scalability

Large-scale datasets are either infeasible or too resource intensive to be represented in a single frame on a client. The following are a few common issues relating to large-scale visualzation.

1) Datapoints involved are too large to fit in a single frame on a client's interface. Additionally, representing the entire spatiotemporal domain of the underlying data at the client is not scalable due to the large amount of server-side processing and network communication it would entail, especially in a multi-user environment. As a result, exploratory browsing is commonly utilized where only a fraction of the entire data-space is visualized by users at varying resolutions through a sequence of shifting their viewport over various regions of interest.

2) A single visual representation is often not capable of representing all of the important information contained in high-dimensional data, for instance, identifying regions of interest or important features for further analysis. This is particularly true in case the underlying data is multi-dimensional. For instance, satellite image datasets, such as Sentinel-2 satellite imagery comes in the form of 13-band TIFF files which are not possible to be represented as they are on a client-side 2D visualization interface.

1.1.2 Concurrent Evaluation of Spatiotemporal Queries At-Scale

Our goal is to facilitate users to have concurrent access and visualize our underlying data through analytical spatiotemporal queries. Front-end users can be a non-expert in the field of data science and not experienced in programming and running analytical jobs over big datasets.

They typically interface with the raw data through exploratory analysis. Our servers should be able to adapt to concurrent queries over varying spatiotemporal extents and must be adaptive in case of hotspot scenarios to maintain high throughput of these query evaluations.

To ensure sustained attention from these users, we need to enable smooth and flexible visualizations against each user actions over the front-end UI. Additionally, queries from individual users may be duplicates or have significant overlaps between successive requests, which is a common characteristic for spatiotemporal data access. We need to effectively identify such duplicate/redundant accesses to avoid re-computation as much as possible to alleviate stress on the distributed server.

1.1.3 Hotspots

Access logs of GIS systems have shown that geospatial data is accessed with spatiotemporal locality [9]. For a distributed web-based service in a cloud-based environment, we need to handle multiple users with simultaneous data access requests to the server. This leads to majority of the user accesses being limited to a small section of the overall spatiotemporal data-domain, which can lead to frequent hotspots/bottleneck over a small number of nodes over our distributed server, if not handled properly. These hotspots adversely affect query latency and hence interactivity of the applications.

1.1.4 Data Consistency for Dynamic Datasets

In a dynamic storage system, quality of query results might degrade over time in light of new data. This could introduce error in the query results. This loss in data consistency could be either due to cached results becoming stale over time or accuracy of models built over old data decreasing with newly added data. Our system needs to be able to track its accuracy over time. Additionally, depending on the user requirement, we provide tunable data consistency guarantees for our various query operations to maintain good perceptual quality of the visualization results by keeping the accuracy of the results withing a desirable threshold.

1.1.5 Analytics Over High-dimensional Datasets

Voluminous high-dimensional datasets have become common over the years and are available in the form of satellite imagery, or spectral imaging from biology and astronomical observations. Hence, with data getting larger and more complex (multi-dimensional) queries over such data to derive analytical information entails resource-intensive computational methods. Ensuring interactivity for analytical processes introduces an additional challenge relating to the size of the individual data objects. For example, in case of high-dimensional heterogeneous datasets, such as satellite imagery, which are available in the form of multi-band rasters, processing relevant images to look for specific phenomenon, such as slope can be a compute intensive process leading to higher response times. Schemes for efficient geoprocessing of such datasets along with extraction of compressed representations of these data objects can help achieve interactive analytics over them. Additionally, these compressed representations must support extraction of analytical information at multiple levels of requested spatiotemporal resolutions, further adding to the complexity of the problem.

1.2 Research Questions

Through this research we aim to address the following research questions:

- **RQ1** *How can we ensure that front-end users are not overwhelmed by the scope of the underlying data?* Our distributed datasets have large spatiotemporal extent, volume and heterogeneity. The majority of the processing tasks are to be handled by the back-end. The front-end UI must be able to interact with the server through a sequence of actions that can be translated into a set of simple data queries. Any related data structures on the user-side should be lightweight and have low memory foot-print.

- **RQ2** *How can we leverage users' access patterns to further improve interactivity of spatiotemporal queries over voluminous datasets?* To improve interactivity of analytical operations, we aim to design our in-memory data-structures to be sensitive to the behavior of

user groups in terms of their access patterns. Geospatial access patterns have been shown to follow spatial and temporal locality [9], which implies that at a specific instant, queries over the entire dataset is focused on small spatiotemporal neighborhood. Our framework aims to effectively leverage these patterns to help improve hit-rates of our in-memory structures.

- **RQ3** *How can we alleviate hotspots that stem from the skewness in user access patterns while ensuring low latency and preserving perceptual quality?* The distribution of geospatial objects' access popularity has been shown to follow Zipf's law [10] where a majority of data-access at a particular instance is focused on a small section of the domain of the dataset. This would lead to transitional hotspots over a small section of our distributed cluster. We enable our framework to be adaptive to such erratic query volumes by implementing, among others, ad-hoc, decentralized replication schemes.

- **RQ4** *How can we estimate results over fast-evolving datasets without compromising on the fidelity of the generated visual artifacts?* We implement several measures to continuously re-evaluates the efficacy of our in-memory framework to ensure that the accuracy of the query results stay within a desired threshold. These measures include bitmaps to constantly check for stale in-memory data objects and periodic evaluation of deep-learning models to evaluate their overall accuracy.

- **RQ5** *How can we quickly extract meaningful insights or performs processing, capable of being rendered on a 2D visual interface, from a multi-dimensional data-store?* We make adjustments to our framework for high-dimensional distributed datasets that enables us to house them in-memory and run high-throughput analytics over them. We avoid resource-intensive geoprocessing of high-dimensional data by implementing model-based analytics over low-dimensional embeddings generated out of these high-dimensional data during ingestion. This also enables significantly larger portion of the overall data to be housed in-memory, thus bringing down the overall disk I/O.

1.3 Overview of Approach

Our approach to enabling interactive spatiotemporal analytics involves the design of 4 interconnected frameworks that can help alleviate unnecessary server-side disk I/O as well as reduce network communication:

- An efficient distributed in-memory framework that is lightweight and agnostic to the underlying distributed data-store.
- An efficient scheme to for rapid identification of relevant cache elements and fast cache maintenance. Additionally, we enable dynamic replication of currently popular cache segments to alleviate hotspots.
- A client-side lightweight deep neural network-based framework capable of regenerating effective data representations from partial in-memory data
- A deep neural network-based model capable of extracting low-dimensional embeddings, followed by models trained to extract insights from such embeddings.

In order to facilitate effective data retrievals that are aligned with the needs of visual exploration we design a distributed, dynamic caching scheme (**STASH [11]**) to alleviate I/O overheads associated with disk accesses. We facilitate rapid data discovery minimizing query-forwarding between the cluster nodes and local traversals within the data structure. STASH employs a dynamic data caching and query orchestration scheme that accounts for spatiotemporal locality of users' visual navigational patterns, especially those prevalent in dominant geospatial explorations, thus ensuring that it retains the data fragments with higher probability of access in the near future. We also design a dynamic load balancing mechanism for our in-memory framework that scales with hotspots stemming from increased interest over a small spatiotemporal region and simultaneous user accesses.

Our in-memory framework is lightweight and can also be installed on a single Client-side node to help identify and avoid duplicate or overlapping queries and precludes excessive data retrievals

and transfers from frequent view changes triggered by a user. Additionally, we couple client-side Stash with additional trained progressively trained [12] deep GAN-based networks (called **Glance** [13]), allowing it to further re-purpose low-resolution data fragments for accurate high-resolution representations, bypassing queries to the server. We ensure that GlanceNet models are space-efficient and include a Glance Error Mitigation Module (GEMM) that informs STASH of potential errors involved with a requested spatiotemporal region. At each level, GlanceNet models quadruples the resolution of the image.

We plan to further adapt our in-memory data structure for voluminous high-dimensional datasets, such as satellite imagery, where each data object is a multi-spectral image. In such cases, operations such as identifying regions of interest or determining composition of the geospatial region, would require additional image transformations and processing for each relevant image tile in the distributed cluster. To reduce the disk and processing time involved for these operations, we plan on implementing a model to extract low-dimensional memory-resident embeddings that are versatile and can be used to extract relevant information relating to the multi-spectral tile that they represent.

1.4 Contributions

This research presents the following novel contributions aimed at scalable exploratory analytics over spatiotemporal data, with an supplementary focus on high-dimensional spatiotemporal datasets:

1. A distributed, in-memory cache for hierarchical aggregation and query evaluations [11, 14]. This graph-based cache acts as a middleware over a distributed file system. Its contextual caching mechanism is tailored to leverage spatiotemporal query patterns for fast exploratory analytics by caching high-priority data objects from past query results based on their frequency and freshness. This helps avoid expensive disk I/O, redundant computations, and network usage at the server.

2. An adaptive and decentralized load balancing replication scheme to handle any hotspot that might result from erratic distribution in user requests due to the spatial and temporal locality of their access patterns. This scheme helps to further improve the throughput of our in-memory cache in hotspot scenarios [11].

3. A set of frameworks [13, 15] for the extraction of latent or summarized representations for voluminous spatiotemporal datasets with high-dimensional data objects. These representations can act as surrogates for real data, to be used during query evaluations without any need for disk I/O, thus facilitating rapid analytics.

4. Building of deep neural networks for extraction of relevant information from these partial/latent representations with high accuracy, reducing the need for exhaustive geoprocessing. Our training leverages various concepts such as transfer learning and multi-task learning to support more stability and fast turn-around times for our model training [16, 17].

5. A set of error handling/ mitigation schemes to address the challenge of regional characteristics of deep learning models over large spatiotemporal data domain [18–20]. These schemes aim to mitigate model inaccuracies resulting from regional variations, ensuring more robust and reliable data analytics. Additionally they include schemes for rapid and stable simultaneous training of multiple deep learning models in a distributed setting in a resource-efficient fashion [21–23].

1.5 Dissertation Organization

The rest of this dissertation is organized as follows: Chapter 2 explores the background and related literature, contrasts with our approach, and analyzes the areas that are not addressed or considered by the current state of the art. Next, we provide an in-depth overview of our methodology in Chapter 3, followed by an overview of our frameworks and brief discussion on various components of our framework targeted at interactive analytics in Chapter 4. Next we describe the individual components of our framework starting with our distributed in-memory framework,

STASH and its various components in Chapter 5. Following this, we go into detailed description of our various approaches to generate compressed/ latent representations of high-dimensional data objects for voluminous datastores - in Chapter 6 we describe the extraction of low-resolution/ latent embeddings of satellite images, and in Chapter 7, we discuss our methodology of maintaining multi-resolution online summarized representation of the data in a hierarchy of non-overlapping spatiotemporal extents. We then discuss our model-driven extraction of insights from these compressed representations with Chapter 8 outlining the framework GLANCE that uses in-memory low-resolution data to generate high-resolution counterparts *in-situ*, and Chapter 9 discussing the use of multi-task learning in our framework ARGUS to leverage in-memory latent representations for extraction of insights instead of the actual data objects with high fidelity. Chapter 10 discusses our various approaches to implementing transfer learning for training of multiple models parallely in a distributed setting while minimizing the amount of network I/O and computational load, followed by Chapter 11 with our conclusions.

Chapter 2

Background and Related Work

With data being collected at an unprecedented rate, the number of big data available for analysis has seen a drastic increase in the past few years. The recent popularity in the area of Big Spatial Data has seen an increase in the number of such spatiotemporal datasets, along with technologies designed specifically for spatiotemporal data management, data processing, and spatial analysis (such as spatial query, visualization etc) [24]. Distributed analytics over large-scale datasets has been handled thorough parallel processing over a cluster of commodity nodes by a variety of frameworks such as Hadoop [25] and Spark [26]. However, interactivity over such voluminous datasets has proven to be a challenge.

An exemplar of a DHT-based system that is aligned with the spatiotemporal characteristics of the data is Galileo [27, 28]. The Galileo system includes support for geometry constrained queries [29, 30], ad hoc query [31], and analytic queries [32, 33] with support cloud bursting [34]. The crux in Galileo is on-disk storage of spatiotemporal data and while the system has been used for visualization, it has not been used in the context of high-dimensional visualizations as envisaged in this work. Scaling is such systems are based on leveraging virtual machines on the server side [35, 36]; this will targets the effective use of caches both at the client-side and server-side to distribute workloads and ensure interactive visualizations.

2.1 Need for In-memory Analytics

Apache Hadoop is one of the most widely-known distributed cluster-computing framework. Hadoop's MapReduce framework enables the distribution of large data into smaller partitions called chunks and parallel execution of analytics over those chunks, followed by accumulation of the results. One of the main advantages of using the MapReduce framework is its ease of use and can be considered a good solution if the speed of processing is not critical. However, Hadoop does not fare so well in case there is a sequence of operations involved or an iterative algorithm

due to the storing and then reloading of intermediate results between iterations to and from disk, respectively. This leads to large disk I/O and brings down the query speed by a considerable amount.

Apache Spark counters the issue of Hadoop MapReduce's high disk I/O by allowing the data partitions, known as Resilient Distributed Datasets (RDD) [37] to be loaded in-memory. RDDs are fault-tolerant, read-only partitioned collection of records that can be persisted in both in-memory and on disk, based on the available memory and user specification. One of the main features of an RDD is its coarse grained fault tolerance scheme where a lineage graph is associated with each RDD, that helps in the regeneration of an RDD partition from any checkpoint in case of a loss or failure. RDDs allow users the option to specify their partitioning scheme and the persistence scheme. The partitioning scheme determines how the data will get partitioned around the cluster and persistence scheme helps users specify a suitable storage strategy for the RDD. A well-defined partitioning scheme helps in optimizing the parallelism of the RDDs over the cluster and hence helps reduce latency. Spark's in-memory computation and persistence schemes allow direct fetch of data-elements between successive operations leading to a 100 fold improvement in latency for queries compared to that of Hadoop. Thus, in-memory staging of data-elements is key to ensure fast and interactive analytics.

Although Spark provides an important feature of in-memory loading of data partitions, there is no way to identify relative importance of the data partitions that are in-memory. This is why, in case of an overflow, Spark automatically monitors cache usage and drops old data partitions in a least-recently-used (LRU) fashion. This is not an efficient partition removal scheme, especially in case of iterative analytics, where we have seen that either the working set, i.e. candidate partitions that are more likely to be accessed can be deduced from the previous partial solution itself, or in the case of exploratory analytics, where the partitions' relative importance can be gauged by the frequency and recency of access, along with clients' pattern of actions.

2.2 Scalable In-memory Analytical Frameworks

Several frameworks have been designed to support scalable visual analytics with their own adjustments to support efficient evaluations. Forecache [38] proposes a prefetching scheme that predicts the data-tiles to be queried in the future based on user's past behavior and recent movements to improve latency. Similar work involving pre-computations have been suggested in [39–43]. In imMens [39], multivariate data tiles are precomputed to provide scalable panning and zooming as done in Google Maps [43]. [40] uses a data cube structure which stores all possible precomputed aggregations at multiple levels of resolutions over the database. However, the system does not scale with dataset size as it requires the entire cube structure in-memory. HashedCubes [41] is built on a pivot/array scheme that maintains a partial ordering scheme for all possible dimensions. This system becomes more memory efficient than Nanocubes [40] by avoiding precomputations of aggregations and using the sorted arrays to compute aggregations on-the-fly. However, this results in longer query evaluation time. Also, the system is not compatible with streaming data, requiring the generation of pivot arrays every time as new data arrives. Kyrix [42] provides a generic visual data explorations system. Another system used for fast data visualization is Tableau [44], which can connect to a variety of underlying databases and support spatiotemporal queries over tabular data [45].

There are several existing data storage systems that can support analytics. SciDB [46] is a column-oriented DBMS designed for multidimensional data management and analytics and provides support for complex analytics over multidimensional data. Storing geospatial data is also supported by PostGIS [47] by adding support to PostgreSQL, an object-relational database. Systems such as Spark [26] and Shark [48] load datasets in memory in a distributed fashion and then allow further analytics. BlinkDB [49] is built on top of Spark is a parallel, approximate query engine which provides interactive SQL queries on stratified samples of large volumes of data.

Load-balancing and cache replacement problems have been explored in papers [50–54]. Paul et al. [52] describe a centralized control architecture for distributed coordinated caches to provide better web access times. Other work has proposed a load-balancing method that considers both

localized access control and balanced load allocation [50]. This leverages a static caching approach that assigns hot-spotted data to the server with higher processing power. The system also includes queuing theory and cache distribution strategy to achieve optimum processing time for data requests. However, it doesn't incorporate a cache replacement scheme to deal with changing hotspots. In [51], a replication strategy based on access characteristics of geospatial data is suggested to provide a high-speed caching system. They propose the use of classic Q-value scheme to allocate cache by replicating hotspot tiles to multiple servers resulting in more replicas for more popular tiles. The paper aims at balancing the utilization rate of each caching server by assigning replicas according to the processing abilities of each server. Although, this can result in the fast depletion of the cache and may cause more delays due to frequent cache replacements.

Since disk I/O is significantly more time-consuming than in-memory operations, loading data objects in-memory is a common strategy to reduce latency. Loading pre-computed aggregations of tiles into memory has been suggested in [39–41]. However, this is not a feasible strategy if memory space is limited. Additionally, these systems are not compatible with fast-evolving datasets, requiring re-computation every time as new data arrives.

2.3 Exploratory Visual Analytics

Visual scalability in voluminous datasets precludes visualization over a large section over the overall data-domain. *Exploratory browsing* is used as an alternative mode of visualization in such cases. Here, since the users view the entire dataset through a limited viewport, initially, they are not sure of exactly the data-segment they are looking for. Rather, they form an opinion or hypothesis about the data or uncover patterns of interest through traversing a given dataset over varying extent and resolutions through a visualization interface. This kind of analytics is common in visualization systems (e.g., Tableau [55], QlikView [56], etc.), where users access different views of the dataset through a visualization interface allowing scientists to explore the data at "rates resonant with the pace of human thought" [3,4]. Through this kind of repetitive movement around the data-space, the user, which can be a data-scientist or a naive user (navigating a visualization interface like Google

Map [43]), performs actions, such as panning, zooming, drill-down, roll-up (OLAP actions [57]), which get converted by the interface to a query understandable by the back-end analytics engine. Commonly, such browsing involves exploring different portions of the data-space at a coarser level (through actions like panning), identifying regions of interest and then zooming in for further details [58].

At the end of each action, the user investigates the output on the interface, gains more knowledge and insight of the data and based on that knowledge, performs the next action. The entire cyclical process of query and knowledge gain is shown in figure Fig.2.1. Also, there may be multiple users accessing similar regions, as part of a collaborative effort, which leaves scope for specialized strategies for avoiding overlapping queries.

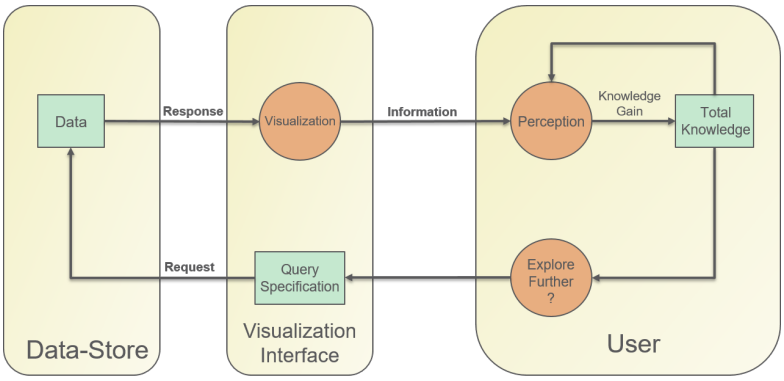


Figure (2.1) Exploratory Browsing Model

2.4 Latency vs User Experience

In exploratory browsing, slow response times at the front-end can limit the range and depth of the investigation from the user. Lack of fluidity in between successive actions can be detrimental to the attention span of a user in these cases. User experience in exploratory systems deteriorates with latency at orders of 10 [8]. At latencies below 0.1 seconds, users feel like their actions are directly causing change at the interface, whereas, for sub-second latencies, i.e. latency below 1

second and above 0.1 seconds, although users notice the short delay, they can "stay focused on their current train of thought". Beyond that limit, the attention span of a user gets hampered and continued delay will lead to disengagement.

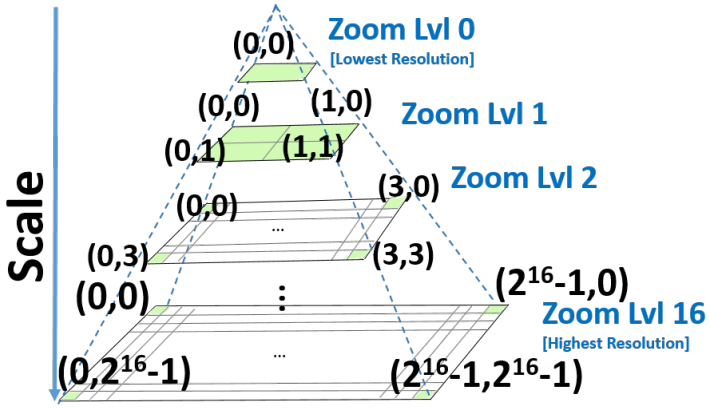


Figure (2.2) Tiles Layers

2.5 Locality of Access Patterns

Users' access patterns for geospatial data has the tendency to follow the principles of spatial and temporal locality [51].

2.5.1 Spatial Locality of Access

This refers to the property that at any instant, user requests are highly concentrated over small clusters of the entire spatiotemporal extent of the data. This means that if a spatiotemporal data region was accessed by a user request at time t, the adjacent spatiotemporal regions, too, have a high probability of being accessed at that instance of time and that probability reduces as we move away from that region.

2.5.2 Temporal Locality of Access

Temporal locality is the property that if a spatiotemporal region was accessed by a user request, there is a high probability for it to be accessed again in the near future. This kind of user access pattern is in line with *Zipf's Law* [59], which can be expressed as:

$$p_i = \theta / i^{\alpha}$$

where p_i represents the probability that a spatiotemporal region of i^{th} popularity will be accessed again and θ is a constant and α is the Zipf parameter. The equation above implies that the more frequently a region is accessed, the higher is its probability of being accessed in the near future.

2.6 Tile layers

Tile layers [60](Fig. 2.2), a widely used data-structure in visual analytics, comprise a set of pre-computed, partitioned, and rasterized data tiles, stored on a server, that are fetched in groups through user queries [61]. Visual analytics applications often rely on data aggregation and sampling techniques over in-memory caches to improve interactivity - organizing data in the form of tile layers can help in fast query evaluations for aggregate queries.

Sketching algorithms have been used in the context of spatiotemporal data and streaming environments [**?**, 62] to reduce their memory footprint for in-memory storage. Similarly compact data representations have been explored in the context of grid computing environments [63, 64]. Often such systems are backed by streaming infrastructure [65–67] to ingest data effectively. However, these approaches have not targeted real-time visualizations of spatiotemporally evolving phenomena.

Model building using deep neural networks for multiple upscale factors using low-resolution/ compressed in-memory counterparts is also an option. Howver, if done in an isolated manner, training such models can lead to deeper layers and larger convergence time. To circumvent this,

we harness progressive training of GANs [12, 68, 69] where we train models for incrementally higher upscale factors by repurposing a trained lower-scale model.

2.7 Progressive Learning

Adversarial learning using Generative Adversarial Networks**(GAN)** [70] have produced realistic-looking images for **Single Image Super Resolution (SISR)** [68, 71, 72] problems. GAN learns through a minimax game between a generative and a discriminator network [73]. Achieving HR image with higher upscaling factor is challenging because a significantly down-sampled image cannot preserve the crucial high-frequency information [74], resulting in blurry images. There are several approaches to improve the perceptual quality of the super-resolved satellite images [74–76].

Performing SISR for tiles over multiple resolutions, requires training models to should support multiple upscaling factors regardless of the zoom level used for the input. Most existing SISR implementations target only a single upscaling factor. This increases the complexity of model training; each model with different upscaling factors should be trained separately from scratch and results in prolonged training times. Incremental training of GANs [69] provides an alternative approach by leveraging progressive GANs [12, 77] that can achieve upscaling factors of up to x8. The process trains models in a nested fashion by training simpler models and upon convergence appending an extra set of dense layers to the trained a more complex model. We leverage this incremental curriculum learning approach – leveraging knowledge gained from a lower-level SR network to train for higher SR in our models to simplify the higher upscaling problems and speed-up the learning process.

2.8 Multi-task Learning

Multi-task learning (MTL) [78] is an effective modeling technique where multiple models learn related tasks jointly to support a mutual exchange of knowledge that facilitates generalization. MTL allows the model to learn shared representations between tasks, which can lead to more efficient and effective learning [79]. Feature-based multi-task learning aims at learning common

features through generalization among related tasks as a way to exchange common knowledge. Multi-task learning involves training of machine learning models with data from multiple tasks simultaneously, using shared representations. This enables the models to acquire shared knowledge between a set of related tasks and also improve its robustness by assimilating knowledge across multiple domains. These shared representations increase data efficiency and can potentially yield faster learning speed for related or downstream tasks, helping to alleviate the well-known weaknesses of deep learning: large-scale data requirements and computational demand. Additionally, MTL can also reduce the amount of data needed to train a model, as the model can use data from one task to improve performance on another task [80].

MTL has been successfully applied to the problem of object detection and semantic segmentation through models such as Faster R-CNN [81] and Masked R-CNN [82], where related tasks like object boundary detection and image segmentation are trained collaboratively through a shared trunk (backbone) followed by branched heads for downstream individual tasks. A potential pitfall of this approach is that training multiple models jointly can be both compute and resource-intensive since all the model layers combined have to be optimized simultaneously. This is especially true for deep learning models.

Chapter 3

Methodology

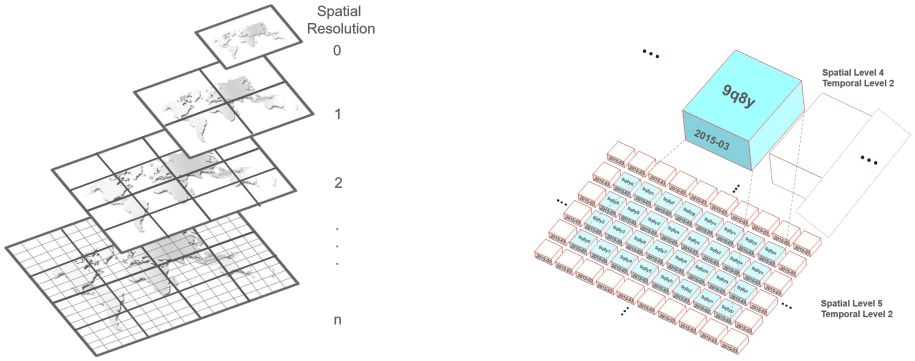
In this section, we describe the various components of our framework that help enable interactive query evaluations over spatiotemporal data. The central component of this framework is a distributed in-memory data-structure, STASH, that uses a maintenance and replication scheme that is influenced by users' access patterns. We couple this with additional deep neural-network modules to help - 1) bypass unnecessary disk I/O and/or client-server communications by utilizing partial in-memory data fragments to generate requested ones and 2) to generate low-dimensional representations of the underlying data-objects capable of being analyzed or processed in case the underlying dataset consists of large data objects, such as multi-spectral imagery. This adjustment can help increase the capacity of our in-memory to house relevant data segments and evaluate them faster during queries.

3.1 In-memory Hierarchical Metadata Graph (RQ1, RQ2, RQ3)

STASH's data model is designed to efficiently store previously generated summary results in-memory from past queries and reuse them in case similar queries are performed by users in the future. The goal is to ensure the most relevant data segments that are likely to be accessed in the near future are retained in-memory as the in-memory storage filled over time.

STASH [11] is logically organized as a multi-relational property graph with data that is aggregated at multiple levels of spatiotemporal resolutions. The data in STASH is stored in the form of a collection of identifiable blocks or chunks with specific spatiotemporal bounds (we call them Cells) that can be rummaged and reused from the in-memory store. Organizing data based on the spatio-temporal resolution at which they are accessed allows us to support fast evaluation of aggregate queries at multiple combinations of spatial and temporal resolutions over our datasets.

STASH can work in conjunction with any distributed hash-table(DHT) [83] storage system and adopts its partitioning scheme to facilitate data locality between in-memory fragments and their



(a) **(b)**

Figure (3.1) Overview of hierarchical organization of Cells in STASH.

actual on-disk counterpart. Each node in this graph (called **Cell**) represents results of data aggregation for a fixed, non-overlapping spatiotemporal bound and resolution and are labeled by their spatial and temporal information as attributes. Vertices are organized in a hierarchical fashion based on their resolution and those with the same spatiotemporal resolution are grouped at the same depth/level. Fig.3.1 shows a 2-dimensional representation of the hierarchical relationship between STASH Cells and the nested spatiotemporal bounds between hierarchically related Cells. Identification of neighboring and hierarchical (parent or children) Cells in STASH is straightforward and can be directly evaluated using the properties of the spatial and temporal partitioning scheme.

STASH employs maintenance scheme influenced by the spatial and temporal locality of access patterns, when spatiotemporal region gets requested to ensure data Cells most relevant to future interest are retained in case of memory overflow. Additionally individual nodes in STASH adapt to potential hotspots in a dynamic and decentralized fashion. Nodes continuously monitor the pending requests queue size on a cluster node and on crossing a threshold, initiate a *Clique Handoff* for the most active spatiotemporal regions on it, which are grouped in terms of **Cliques**. Cliques get replicated at *antipode* nodes, which are nodes whose spatiotemporal domain is diagonally opposite to the current node, since it is the least likely to become hotspotted anytime soon.

3.2 Model-Driven Data Reconstruction (Glance)(RQ4)

On clients, STASH can help further reduce unnecessary communication with the server, using a set of model based on progressively trained Generative Adversarial Networks, called Glance [13], that dynamically reconstructs high-resolution data during zoom-in operations using in-memory historical low-resolution rasters and is space-efficient to facilitate memory-residency at the clients. These modules are trained in a distributed manner at the server and downloaded by the clients.

Glance's GAN-based image reconstruction modules (GlanceNet) adopt progressive growing of GANs [75] to train in an incremental manner in a bid to build more stable models with relatively simpler layers and to converge faster. GlanceNet models' architecture is a combination of single image super-resolution networks and image inpainting networks. The image inpainting portion of the network can help further refine images for which sufficient partial high-resolution data fragments are available.

Additionally, Glance includes the GEMM module which is evaluated simultaneously with GlanceNet training. The GEMM module serves a dual purpose. At the server, it is used to continually keep track of the viability of the GlanceNet modules in light of continuously evolving data-store. On clients, it is used to gauge the model's accuracy over a specific spatiotemporal region, which can be used at the client to determine whether to opt for image super-resolution at the client or perform physical fetch from the server.

3.3 Extraction of Insights from High Dimensional Data (RQ5)

Distributed datasets of high-dimensional data introduce a new set of challenges to interactive visualization. First, the size of each individual data object makes it difficult to house sufficient amount of data fragments in our in-memory data structure (STASH). Second, processing these high-dimensional data to extract suitable insights is time-consuming and resource intensive. One such example is the computation of Normalized Difference Vegetation Index (NDVI) from different frequency bands in multispectral image TIFFs.

Our approach to handling this issue is to perform model-driven generation of a unified low-dimensional representation for these large data-objects. We leverage these low-dimensional embeddings (for each dataset) and train multiple neural network models designed for extraction specific type of insights. Our goal is to demonstrate that the latency for model-based processing of these low-dimensional representations will be significantly lower than actual processing of spatiotemporal objects, which if trained properly, should be achievable.

We plan to train these multiple machine learning models in a collaborative fashion through a multi-task module. This would enable us to fine-tune the same set of embeddings to be optimized for each multiple models. The main challenge of this architecture is that efficient training of multiple model branches jointly through multi-task learning can be compute and resource-intensive since model layers for each of these models combined have to be optimized simultaneously.

3.4 Building Multiple Regional Models At Scale (RQ5)

Models designed over large spatiotemporal extents tents to have regional characteristics in terms of their performance. This research outlines methodology aimed at reducing the computational resource requirements of training deep learning models on spatial data. The methodology is based on transfer learning [80], which involves exhaustively training a small subset of the regional models as a starting point for training the majority of the remainder regional models. These regional models are then fine-tuned using transfer learning to account for regional variations.

Instead of training each constituent model instance from scratch (cold-start), our methodology ensures that model instances have a superior starting point (warm-start) for their weight vectors and coefficients for faster convergence. We have devised three different transfer learning schemes, amalgamated, partitioned, and hybrid to choose from based on the spatiotemporal correlations in the overall dataset. Our methodology has been shown to reduce the completion times for training deep learning models over spatial data at scale by up to 5.3x without sacrificing on the accuracy of the models.

Chapter 4

System Overview

This dissertation explores various supplementary frameworks designed on top of a DHT-based storage system, which when deployed in tandem can help significantly alleviate computational, disk I/O, and network costs over the system. We provide an overview of each of these supplementary components of our system as demonstrated in Fig.4.1 and individually explain how they assist in query analytics in their own specific way.

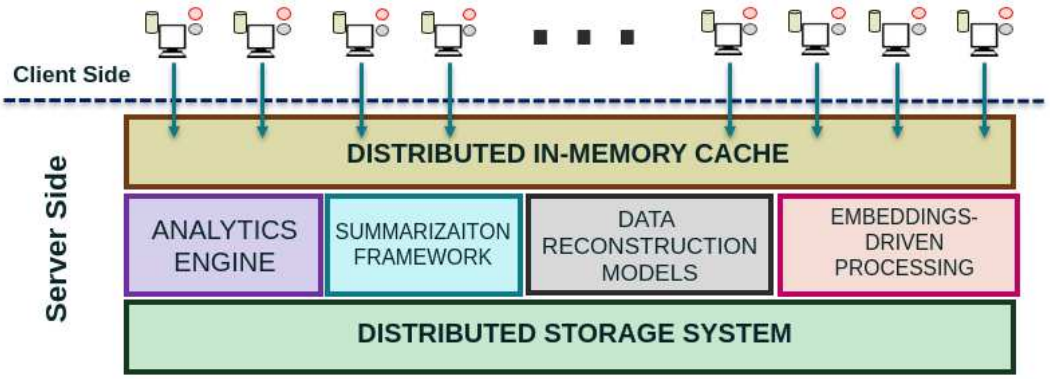


Figure (4.1) System Architecture

4.1 Front-end Visualization UI

In order to facilitate exploratory browsing, the front-end user interface is usually kept lightweight with two key tasks: (1) translate a user action (e.g. panning, dicing, etc.) into a spatiotemporal query over the data storage system and (2) processing the server response to extract a visual representation of the returned summary statistics (e.g., a heatmap or histogram) over the visualization interface.

Our framework can act in conjunction with any visualization engine that is capable of parsing and displaying summarization responses in JSON. In our system, the front-end visualization (as in Fig.4.2) is performed using Grafana [84]. Grafana is an open-source visualization and metric-analysis tool and we have used its WorldMap panel to display spatiotemporal results. Additionally, we have also explored browser-based front-ends designed using React [85] or Flask [86].

4.2 Distributed In-Memory Aggregation Framework

A lightweight front-end means offloading the majority of the processing workload to the back-end servers. In a multi-user system, this would lead to a large number of potentially simultaneous requests from a multitude of users. For each such request, the system has to identify the spatiotemporal subdomain of the dataset relevant to the query parameters, summarise over that segment at the desired resolution, and return a summary report over the spatiotemporal query region.

Our in-memory data storage framework (STASH) acts as an intermediary between the users interacting with a lightweight front-end UI and a back-end distributed storage and analytics engine. Each request over our back-end storage, gets first intercepted by STASH to look for already present in-memory data fragments and the query is modified to search for only the missing spatiotemporal domain. STASH's maintenance scheme improves hit-rate by ensuring that the most relevant data segments stay in-memory. This helps reduce redundant disk I/O, processing and network communication and helps release a significant amount of computational load at the server.

A stand-alone version of STASH can be optionally housed on individual clients that can help reduce making redundant spatiotemporal queries to the server in cases of duplicate/ overlapping requests over spatiotemporal regions. This optional improvement can provide significant boost to interactivity, since clients tend to browse over a small spatiotemporal region at a time.

4.3 Deep Learning Models

We have explored two sets of deep learning models that can work in conjunction with STASH to further improve the interactivity of our analytics applications. The models are trained in a distributed manner over our servers and can be downloaded by clients.

The first set of the models, codenamed **Glance**, aims to repurpose low-resolution data available in-memory to generate requested high-resolution data through multiple levels of super-resolution using a set of neural networks trained using the approach of progressive training of GANs [12]. The models are light-weight and can be housed on clients' memory to help circumvent unnecessary client-server communications.

The second set of models is aimed for speeding up queries over datasets with large data objects, such as multi-spectral imagery. The complexity and size of these data objects makes it difficult to house a decent number of them in-memory. Additionally, processing them also takes time and requires additional resources in cases the queries require us to do so, for instance, computing the slope from a raster tif file containing elevation data. In order to reduce the latency of such operation, we aim to design models to create low-dimensional embeddings from the actual data objects from which desired insights can be extracted using deep learning models.

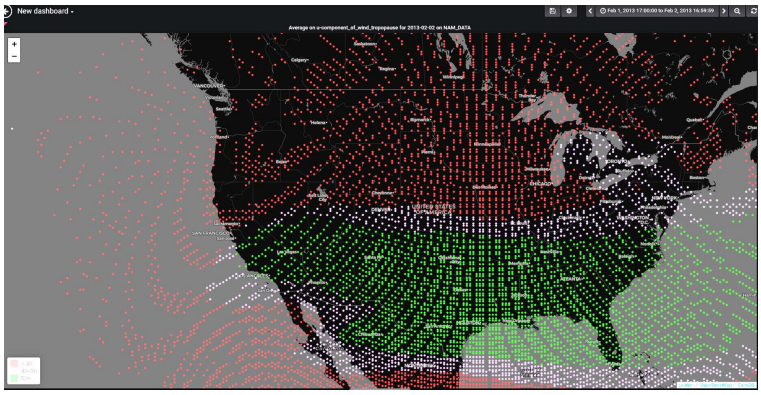


Figure (4.2) Our Front-end Visualization of NAM Data

4.4 Back-end Distributed Storage

Our framework is compatible with any Distributed Hash Table(DHT) [87]. In our implementation, for the back-end storage and analytics engine, we use Galileo [88], a distributed storage and analytics framework for large multidimensional, spatiotemporal datasets. Galileo is a zero-hop Distributed Hash Table(DHT) based storage system that can use various spatial hashing schemes, such as Geohash [89], to generate data partitions that store and colocate geospatially proximate data points. The granularity of the coverage of a data block is determined by the length of hash code managed by the nodes. STASH builds on Galileo's distributed query evaluation capability to efficiently query and then summarize over data points that match a user query at varying spatiotemporal resolutions.

4.5 Multi-Resolution Spatiotemporal Query

Queries for visual applications generate a set of pixel-level aggregations that matches the user's query. For example, the following SQL query shows an aggregation query for rendering maximum temperature values at a given spatial and temporal resolution (*spatial_resolution, temporal_resolution*) over an area and during a period specified in the *Query_Polygon* and *Query_Time*, respectively.

```
select avg(temperature), ...
from Geospatial_Dataset
where coornidates in Query_Polygon
and time_stamp in Query_Time
group by spatial_resolution, temporal_resolution
```

The data objects within a minimum bounding box will be filtered based on the spatial and temporal range and then processed and aggregated on the requested *temporal_resolution* and *spatial_resolution*. These post-query processing or aggregations will be performed only over the data records matching the query. If a user tries to investigate an area with different resolutions, the

query and resolutions should be modified. Therefore, the back-end data system has to evaluate different aggregation queries for almost every user interaction.

Chapter 5

Distributed In-Memory Hierarchical Metadata Graph (STASH)

Here explore the component that forms *the back-bone of our high-speed analytics framework*, **STASH** [11]. It is a designed as a distributed in-memory cache that supports hierarchical aggregation queries for the large-scale spatiotemporal visual exploration. Unlike existing backend storage systems, STASH works in tandem with underlying distributed file systems and provides an in-memory data storage layer that can flexibly scale based on available resources. STASH supports aggregation queries based on the user's navigation patterns such as slicing, dicing, zooming, panning, rolling-up and drilling-down and caches the aggregated results in main memory to provide reusability of the query outputs. STASH also alleviates data retrieval hotspots caused by popular areas-of-interest through a dynamic load-balancing scheme.

In providing exploratory visual analytics to the users over large datasets, computational latency of the back-end server is quite high, especially in multi-user environments. Voluminous datasets, housed over a cluster of nodes, are hard to process interactively, leading to high response time at the front-end UI. While performing analytics over the datasets at varying resolutions reduces the amount of data movement between the server and the user side, it does not reduce the number of records that need to be processed to generate that summary. Through our in-memory distributed framework, we attempt to combat the issue of high query latencies over voluminous spatiotemporal data by saving the most relevant results in-memory and fetching those saved summaries in case a similar query comes in. An efficiently curated in-memory summary storage framework will greatly help reduce the disk I/O and hence the number of records processed for a particular spatiotemporal query in case another query with similar spatiotemporal constraints has already been processed by the system.

Although storing previously queried analytics in-memory will greatly reduce the latency of similar queries in the future, the size of the main memory is limited and hence, the number of summary entries must be limited as well. In order to account for the limitation of memory, STASH is a sparse graph and populated dynamically as new data domains get requested by users. Also, cases of memory overflow, an effective eviction scheme for the entries must be devised to ensure that the entries in-memory are the ones that are most likely to be subject to query in the near future.

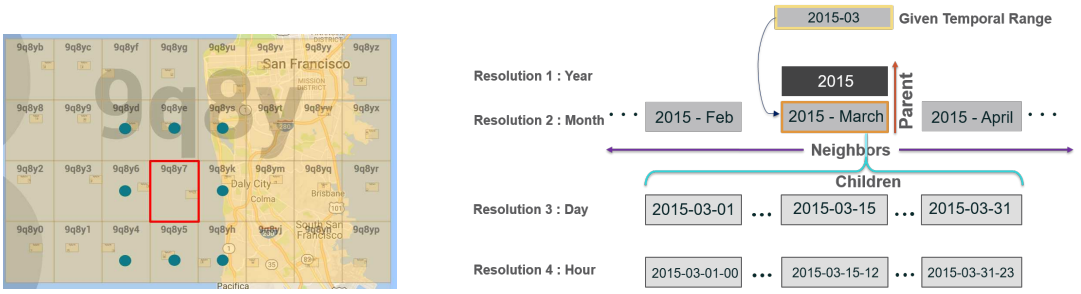
Spatiotemporal datasets are available in a multitude of formats, such as vector (shapefiles, geoJSONs), raster (GeoTIFFs), point data, etc. STASH is amenable to storing aggregate statistics over any of these data formats over a disjoint set of spatiotemporal bounds organized in a hierarchical fashion for efficient storage and query retrieval.

5.0.1 Data Model and Query Evaluation

This section outlines the different components of the STASH distributed graph and how they interact with each other. STASH's data model is designed to efficiently store previously generated summary results in-memory from past queries and reuse them in case similar queries are performed by users in the future. For that, we organize the results' data to be stored in the form of a set of identifiable blocks or chunks with specific spatiotemporal bounds (Cells) that can be rummaged and reused from the in-memory store.

STASH is logically organized as a sparse multi-relational property graph with data aggregated at multiple levels of spatiotemporal resolutions. This graph is defined as $G_{STASH} = (V, \mathbb{E})$, where V is a set of vertices and $\mathbb{E} = \{E_H, E_L\}$ is a family of edge-sets. Vertices are labeled by their spatial and temporal information and have a set of properties represented as attributes. E_H represents the set of hierarchical edges which exists between vertices that are one level apart in the spatiotemporal hierarchy. Edge $e \in E_H$ with a source vertex V_S and a destination vertex V_D indicates that V_S's resolution is one spatial and/or temporal resolution greater than V_D. We can see that the vertices are grouped hierarchically based on their spatiotemporal resolutions. Vertices with the same spatiotemporal resolution will be at the same depth/level in the hierarchy. Therefore, since

V_S has higher precision data segments than V_D, the bounds of V_D fully encloses that of V_S. The set of *lateral edges*, E_L, maintains the proximity of geospatial locations and temporal ranges between vertices in the same level. If there is an edge $e \in E_L$ between two vertices V_i and V_j, these vertices contain data for the two adjacent areas, i.e., their spatiotemporal bounds share boundaries. Each edge-set provides a distinctive traversal function and helps in the spatiotemporal neighborhood discovery for any region of Cells over G_{STASH}.



(a) Spatial **(b)** Temporal

Figure (5.1) STASH Data Model: Spatiotemporal Relationship Among Individual Cells - Red box outlines the spatial and temporal extent our example Cell. Figure shows its spatiotemporal neighbors and parents.

Physically, each *Level* is maintained in the form of a hashmap of Cells, whose key is a combination of their spatial and temporal bounds. The Cell object's SRI can be used to identify the spatiotemporal neighbors (if present) in the Level. The entire STASH graph is essentially a list of such Cell groups where index of a particular Cell-group is the same as its the level id (resolution). Hence, although logically, the STASH Cells are structured in the form of a graph, they are physically stored in the form of a list of Cells, with each Cell not storing actual edges/pointers to its spatiotemporal relatives, but rather their information/clues that can be used to quickly track them down. The reason for not storing actual pointers to other Cell objects is because of the large number of spatiotemporal neighbors and children that each Cell can possibly have. To get an understanding, each Cell can spatially have 32 immediate spatial children (in case of Geohash-based spatial partitioning) and temporally up to 365, which combined would result in a large number of pointers in-memory.

Table (5.1) STASH Cell Components

Content	Description	
Summary Data	Summary Statistics over datapoints lying in the spatiotemporal bounds of this cell	
Spatiotemporal Bounds	Spatiotemporal bounds of given cell	Geohash
		Time interval (eg. '2015-03')
Spatiotemporal Relationship Information (SRI)	Information identifying spatiotemporal parents, neighbors and children of current cell	Spatial Parent(s), Temporal Parent(s) $\in E_H$
		Spatial Neighbors, Temporal Neighbors $\in E_L$
		Spatial Children, Temporal Children $\in E_H$

5.0.2 Vertex: The STASH Cell

Requesting data at specified resolution involves grouping data points into bins of equal spatiotemporal extents and generating aggregated values for each attribute for all the points that fall within a certain bin. The array of aggregated attribute values for each bin is referred to as a **Cell**. A Cell is the minimum unit of data storage in STASH and represents a vertex of G_{STASH}.

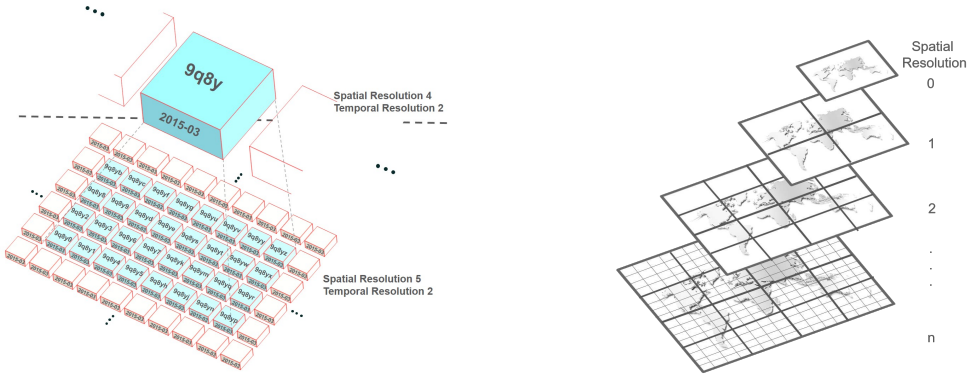
Properties of a Cell

Each STASH Cell, C_i, contains three main properties: (a) spatiotemporal labels, (b) aggregated summary statistics and (c) edge information($\{E_{H_i}, E_{L_i}\}|E_{H_i} \in E_H$ and $E_{L_i} \in E_L$), as shown in Table 5.1. The *summary statistics* are the main content of a Cell and is the information returned to a client program in response to a spatiotemporal request. The spatiotemporal labels describe the scope including the spatial bounding box encoded as Geohash value and the chronological range for the observations. The edge information keeps the Cell aware of its immediate spatiotemporal neighborhood.

Nested Coverage

STASH *Cells* can be thought of as 3-dimensional cubes in the spatiotemporal space, with a fixed bounds marked by their spatiotemporal labels. Cells connected by a hierarchical edge have

nested spatiotemporal bounds, i.e., the bounds of the lower resolution Cell fully encloses the higher resolution Cell. Fig.5.2b shows a 2-dimensional representation of the nested bounds of Cells in the hierarchy. A Cell's spatiotemporal extent is inversely proportional to its spatiotemporal resolution. With every increase in spatial or temporal resolution, a single parent Cell gets broken into a fixed number of child Cells. For instance, Geohashes represent a hierarchy of successively higher-resolution spatial bounding boxes using a string of Base32 characters. So one spatial resolution increase splits each lower-resolution Cell into 32 equally-sized smaller Cells.



(a) **(b)**

Figure (5.2) Spatiotemporal Hierarchical Positioning of Cells

5.0.3 Edge: The Inter-Cell Relationship

STASH maintains two edge sets to represent distinctive relations between Cells. The hierarchical edges represent the spatiotemporal parent(s)/children of each Cell (see Table 5.1). Each Cell can have 3 different parent precisions (one step lower spatial precision, one step lower temporal precision and one step lower spatiotemporal precision). This refinement is applicable to the children nodes as well. The lateral edges help identify the spatiotemporal neighborhood of a Cell.

As shown in Fig.5.1 and Fig.5.2a, a Cell covering a geohash *9q8y7* and time *2015-03* has a spatial resolution of 5 (length of the Geohash) and temporal resolution 'Month'. The Cell has 8 adjacent spatial neighbors - *9q8yd, 9q8ye, 9q8ys, 9q8yk, 9q8yh, 9q8y5, 9q8y4, 9q8y6* (see Fig.5.1a) and 2 temporal neighbors *2015-02* and *2015-04*, which represent its *lateral edge*. Similarly parent-

age and children can be deduced from the Cells spatiotemporal information. For instance, the spatial parent of Geohash region *9q8y7* is 9q8y (spatial resolution of 4) and each Geohash box encloses 32 nested Geohashes, which would represent the spatial children.

5.0.4 Hierarchical Cell Organization

To exploit the lineage and inter-relationship among the Cells, the STASH framework is organized in a hierarchical graph structure to support fast population and updates to the in-memory data. The graph level for a given spatiotemporal resolution is calculated as $(n_j * n_t + n_i)$ where n_s and n_t are the total possible spatial and temporal levels, respectively and n_i and n_j are the spatial and temporal resolution of the current level, respectively. Fig.5.2a gives a view of the relative positioning of two levels with varying resolutions.

5.0.5 Query Evaluation Strategy

STASH's dispersion scheme attempts to maximize data locality by deploying an instance of STASH graph at each node holding data related to that node. Although STASH is organized as a graph, the query evaluation does not rely on traditional graph traversal algorithms that often result in excessive network communications and iterations. STASH provides a set of composable vertex discovery schemes (through the hierarchical and linear edge relationships) that reduce the network communications and iterations significantly. Since the zero-hop DHT maintains the hosting information of the entire key ranges in each node, within a single precision level, the query evaluation requires up to one query forwarding to locate the data. For a STASH cluster with N nodes, the computational cost to locate a Cell within a given precision level is $O(1)$. A *Stash* graph is maintained on each node of the distributed cluster to help avoid disk access on that particular node.

Across multiple precision levels, STASH relies on **precision-level map (PLM)** to check for completeness of the in-memory data. The PLM is a bitmap that associates the Cells contained in-memory for a given level to the actual data blocks in the distributed storage. The PLM is memory-resident and helps identify whether all Cells contained in a given data-block exists in-

memory. Otherwise STASH will consult the PLM to retrieve missing chunks from the data blocks and complete the query evaluation.

Each node, that receives a user query over the subset of data it contains, first checks the STASH hierarchical graph at the query's specified resolution to look for Cells that match the criteria. A data block to *Stash* Cells bitmap lets the system know which data blocks are fully contained in the Cells and which need processing. All missing Cells(if any) are queried from the disk, like it would in case of normal query evaluation and then populated into the graph. Hence, the response would be a combination of the summaries extracted from STASH and the remainder from the file-system.

The missing Cells once processed from the file-system have their SRI information calculated and then populated in the proper spatiotemporal level to which they belong. One thing to note is that, during query evaluation, once the missing Cells have been fetched from the actual file-system, they are combined and sent back as response immediately. The population of these new Cells into STASH happens in a separate background thread.

5.1 Leveraging Visual Navigational Patterns

5.1.1 Proximity Aware Data Dispersion

To preserve Cell relationships, we leverage the strong spatial and temporal locality of access [9] which is characteristic of spatiotemporal user requests. **Spatial locality** implies that if a spatiotemporal region is accessed, its neighborhood also has high chance of future access, while **temporal locality** implies that a region's popularity is directly proportional to its probability of being accessed in the near future, which is in line with *Zipf's Law* [51].

5.1.2 Collective Caching

STASH provides a query optimization scheme tailored for explorative analytics through OLAP operations [57] for multi-dimensional data. A sequence of such operations often involves partially overlapping or nested queries (e.g., a series of panning operators may involve overlapping areas and zooming or rolling-up results in nested queries). Although existing scalable databases support

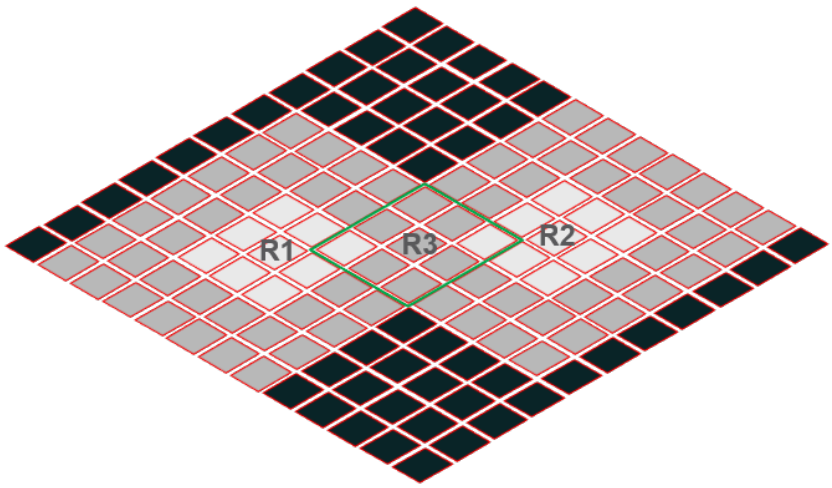


Figure (5.3) STASH Freshness Dispersion Scheme

sophisticated aggregation schemes with caching mechanisms [90], the query output is not reusable by other partially similar queries. STASH's in-memory cache is collectively built through query evaluations from multiple users. Any subsequent query will be evaluated over the cached values first. Disk access is required only if, (a) there are missing values for completing query evaluation, and (b) those missing values are not available by computing from the existing cached values.

5.1.3 Cell Replacement Strategy

We ensure that the spatiotemporal scope of an incoming request has higher probability of overlap with the current cached entries. The total number of possible Cells is much larger than the number of cells that can be persisted in-memory. So for our Cell replacement strategy, rather than focus on the demand of Cells individually by only calculating their frequency of access, we focus more on the spatiotemporal **regions of interest**, i.e. the regions that are experiencing the most user queries at a particular instant, keeping in line with the concepts of spatial and temporal locality of access.

Cell Freshness

The effectiveness of STASH lies in its ability to maintain the most relevant regions in the memory, and to efficiently detect stale Cells and swap them out for more requested regions, in case we reach a threshold due to overpopulation of Cells. In case of a threshold breach, we use the metric of **freshness** to evaluate the importance of a Cell in the STASH Graph. *Freshness* is calculated as the product of the number of accesses to a Cell (updated every time it gets accessed), and a time decay function. Hence, both frequency and recency of access are contributors to the *freshness* of a Cell. Cells in STASH are replaced based on this freshness score.

Freshness Dispersion Scheme

In accordance with the spatial and temporal locality of access patterns, when a request for a spatiotemporal region comes in, we mark both the set of Cells in that region and its immediate spatiotemporal neighborhood as being of future interest, to prepare for possible overlapping requests in the immediate future and update their *freshness*.

Fig.5.3 gives a two-dimensional view of our freshness dispersion scheme at a particular spatiotemporal resolution. Let us say that spatiotemporal regions R_1 and R_2, highlighted by light-colored cells, have been accessed by one or more end-users recently. Temporal locality of access dictates that Cells accessed recently have a higher probability of access in the near future. Therefore, in our freshness dispersion scheme, when regions R_1 and R_2 get accessed, we increase their freshness (by, say, f_{inc}, which is configurable). Also, spatial locality of access suggests that if regions R_1 and R_2 are currently of interest, their spatiotemporal neighborhood will also be a region of interest. To reflect this property, we also disperse a fraction of f_{inc} to the Cells in the immediate neighborhood of R_1 and R_2 (grey cells). This scheme prevents the spatiotemporalneighborhood from being deemed stale, even though they might not have been accessed recently(eg. R_3).

The freshness dispersion scheme described above boosts entire regions that are heavily accessed to be persisted in memory during replacement, instead of disconnected patches that would reflect the actual query areas that were fetched but might hamper the performance and latency

of future queries. STASH Cell replacement occurs in a decentralized fashion on each node and involves evicting the Cells with the lowest freshness score till the capacity goes below a safe limit.

Advantage of the Hierarchical Graph Organization

Updating the freshness of a set of Cells that belong to a user-requested region out of a large collection of in-memory Cells and also to their spatiotemporal neighborhood is a time-consuming operation that can potentially slow down the query evaluation if not done efficiently. The hierarchical organization of STASH Cells allows us to perform the aforementioned functionalities fast and effectively. First, it allows us to isolate the set of Cells that belong to the current spatiotemporal resolution, as well as their parent and child-level resolutions, thus narrowing down the scope of the Cells we have to work with. Second, once the relevant Cells have been identified as part of a queried region, we can easily use their lateral and hierarchical relationship(edges) to find the Cells in their immediate spatiotemporal neighborhood and update their freshness values in a fast manner.

5.2 Autoscaling for High Throughput Query Evaluation

The spatial and temporal locality of user access patterns makes it highly likely for **hotspots** to emerge over the distributed data storage system [91]. A large number of queries focused over a small spatiotemporal portion of the entire data space would lead to only a few nodes servicing a large chunk of queries, leading to a bottleneck. Also, these hotspots are dynamic [92]. In STASH, individual nodes adapt to potential hotspots in a dynamic and decentralized fashion.

5.2.1 Dynamic Clique Replication

Whenever the workload (determined by the pending requests queue size) on a node in the cluster crosses a configurable threshold, that node initiates a **Clique Handoff** for the most active spatiotemporal regions on it. Hotspotted regions in STASH are demarcated in terms of **Cliques**, which represent the most active set of Cells in STASH and acts as the unit of data replication and transfer in STASH. *Clique Handoff* is the decentralized process of the **hotspotted node** finding the most suitable candidate node (called the **helper node**) to house replicas of its *hottest* Cliques.

A helper node maintains two STASH graphs - one local and one guest (containing replicated Cells from other hotspotted node(s)).

Similar to the case of Cell replacement, in replication too, our goal is to ensure that heavily accessed **regions** are replicated, instead of a disjoint patchwork of Cells so that during hotspots, most incoming requests would request fully replicated regions and hence be siphoned off to helper node(s), thus alleviating some of the processing load.

5.2.2 Clique Handoff Process

The Clique Handoff process involves the following steps, as depicted in Fig.5.4:

Top Cliques Calculation

The hotspotted node attempts to find the spatiotemporal regions responsible for majority of its workload in the form of Cliques in its STASH graph with the highest cumulative *freshness*. We define cliques, here, as a subgraph of Cells from the STASH graph of a pre-configured size (depth). For example a clique of depth 2 would consist of a Cell C_i and all its children Cells and their children Cells to calculate their cumulative freshness. Cliques are identified by the spatiotemporal label of their topmost parent Cell. The reason for selecting multiple resolutions through a Clique is to brace for future queries at varying resolutions over the same region.

We set the maximum number of replicable Cells at a time to a preset amount, say N. The hotspotted node searches its STASH graph to find the top K cliques whose cumulative size is $\leq N$. The hierarchical structure of STASH graph makes it efficient to identify the Cells that would be in a given clique. These top K cliques are subject to replication from a hotspotted to helper node(s).

Antipode Node Selection

The candidate for a helper node is calculated separately for each clique. Since hotspots tend to be concentrated in small pockets in the spatiotemporal space, our goal is for clique replicas to be housed on nodes whose domain is the most isolated from the current hotspotted region. In our implementation, we look for a spatiotemporal region that is diametrically on the other side of

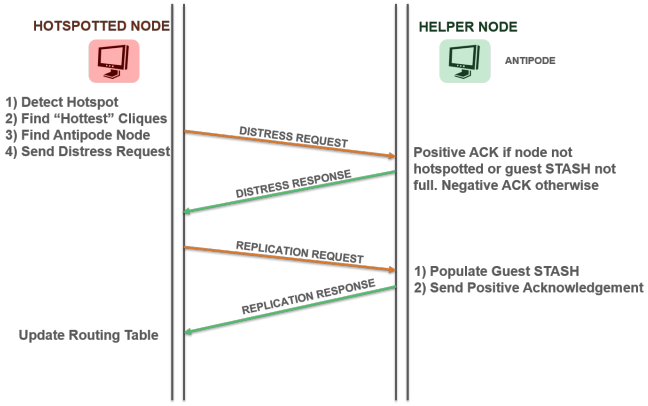


Figure (5.4) Hotspotted Region Handoff

the total spatial scope of the storage cluster. The idea is similar to the concept of *antipode* of a coordinate which is another coordinate on the diametrically opposite side of the globe. We call the node handling this diametrically opposite region the **antipode node**.

We use a Clique's Geohash to find its geohash antipode and then use the DHT's partitioning scheme to identify the antipode node and send it a *Distress Request*. If it is not itself hotspotted and its guest tree can accommodate the incoming Cells, the antipode node sends a positive acknowledgement. In case of a negative acknowledgement, the hotspotted node looks for a region in a random direction around the antipode region and repeats the distress request.

Replication Request/Response

Upon receiving a positive acknowledgement from a helper node, the hotspotted node sends back a *Replication Request* containing the Clique(s) to replicate. On inserting the Clique into its guest STASH graph, the helper sends back a successful Replication Response to the hotspotted node.

Routing Table Population

The hotspotted node maintains a routing table of Cliques that are replicated at helper nodes, along with a bitmap of the actual Cells contained in the clique. This routing table is populated upon receiving a successful Replication Response from a helper node.

5.2.3 Query Evaluation under Hotspot

In a hotspot situation, a user query is first checked against entries in the routing table and if the spatiotemporal region of the user query is found to be fully replicated at another helper node, the user request is probabilistically rerouted to the helper node, thus reducing the load on the hotspotted node. At the helper node, the relevant Cells are fetched from its guest STASH graph, just as it would be from a local STASH graph.

5.2.4 Replication and Cleaning

Each node has a pre-configured cooldown time after hotspot handling. If the hotspot persists after this cooldown time, the Clique Handoff process is repeated and another set of replicas of active Cliques are created on candidate helper nodes.

The guest STASH graph entries also get purged if they are not requested to be persisted within a configurable amount of time. Stale routing-table entries also get purged from the hotspotted node after a pre-configured period signyfying the retreat of hotspot.

Fig.5.5d contrasts the performance of STASH's replication scheme against STASH without dynamic replication under skewed traffic. We simultaneously executed 1000 random county-level requests, centered around a starting region to emulate the hotspot scenario of sudden interest over a single region from multiple users. Our system was configured to initiate Clique handoff with pending requests of over 100. To compare improvement caused by a replication operation, the cooldown time was set high. Fig.5.5d shows the number of responses received each second from the start. We can see that STASH with a dynamic replication scheme processes all tasks ~ 20 seconds before STASH without dynamic replication.

5.3 Empirical Benchmarks

We explore the effectiveness of maintaining of the STASH framework for exploratory analysis over large spatiotemporal data. For that purpose, we profile various types of common OLAP

operations that are characteristic to exploratory browsing and tested the resilience of the framework against high volume of irregular user requests.

5.3.1 Experimental Setup

To evaluate compute-intensive operations with high-density observations, we profiled STASH while performing OLAP operations with spatiotemporal data on a cluster of 120 nodes. Each node in our distributed cluster is an HP Z420 with the configuration: 8-core Xeon E5-2560V2, 16 GB RAM and 1 TB disk. The data is partitioned throughout the cluster uniformly based on the first 2 characters of their Geohash.

To contrast performance with other geospatial caching systems, we have used Elasticsearch [90] on a cluster with 3 master nodes and 120 data nodes. To achieve horizontal scalability and parallelization, the index was split into 600 shards. Three types of caches that were maintained stored the query results, aggregations, and field values on a node.

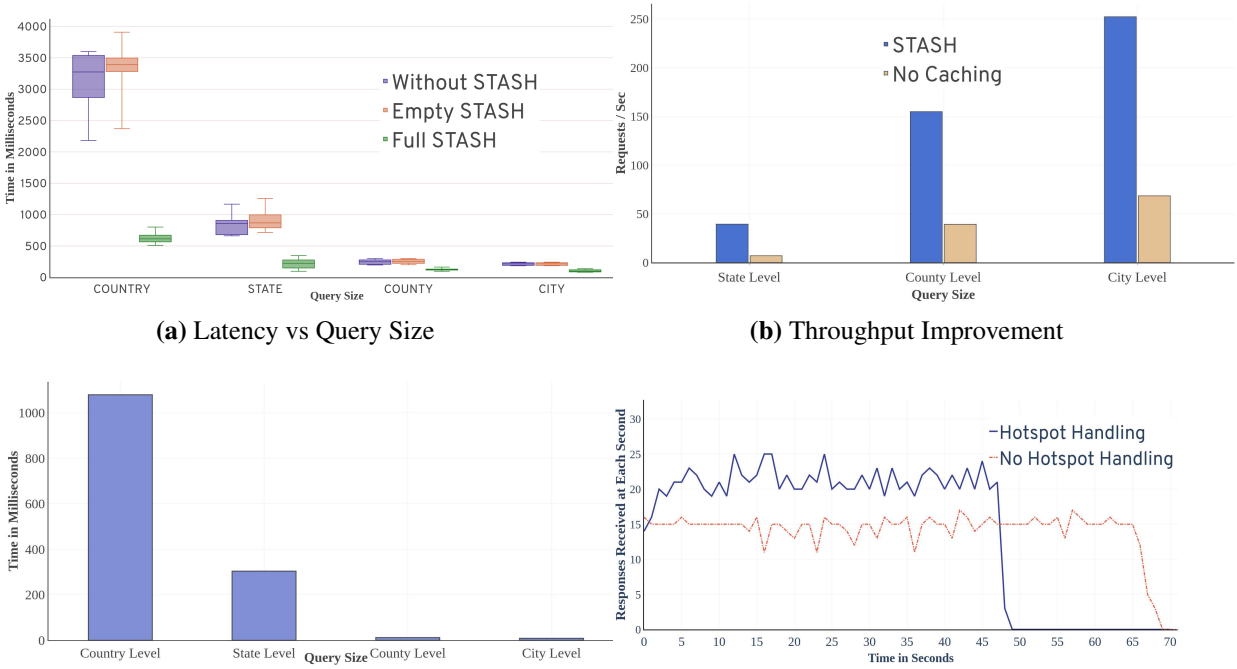
We contrast our query latency with 4 groups of spatiotemporal queries as country, state, county or city level. These represent 4 query sizes that vary in their spatial extent (Query_Polygon) but have a fixed temporal extent which is *2015-02-02* (Query_Time). The spatial extent of the 4 query groups is set using a random rectangle over the data's entire spatial coverage with latitudinal and longitudinal extent of (16°,32°), (4°,8°), (0.6°,1.2°) and (0.2°,0.5°), respectively. The requested spatial and temporal resolutions are 6 and 'Day of the Month', respectively, unless otherwise specified.

The dataset is sourced from the NOAA North American Mesoscale (NAM) Forecast System [93]. The NAM dataset ($\sim$1.1 TB unprocessed) contains atmospheric data collected several times per day for 2013, globally including features like surface temperature, relative humidity, snow and precipitation.

5.3.2 Distributed Query Evaluation Statistics

Query Evaluation Latency

We tested the latency improvement with the STASH framework by evaluating the average latencies of queries of varying sizes for 3 scenarios - the simple Galileo storage system, empty STASH graph with no Cells (worst-case) and STASH graph with all necessary Cells in-memory (best-case - duplicate query). This kind of querying for a subset of the total spatiotemporal extent of the data reflects the *dicing* operation on our system.



(a) Latency vs Query Size

(b) Throughput Improvement

(c) Maintenance Overhead vs Query Size

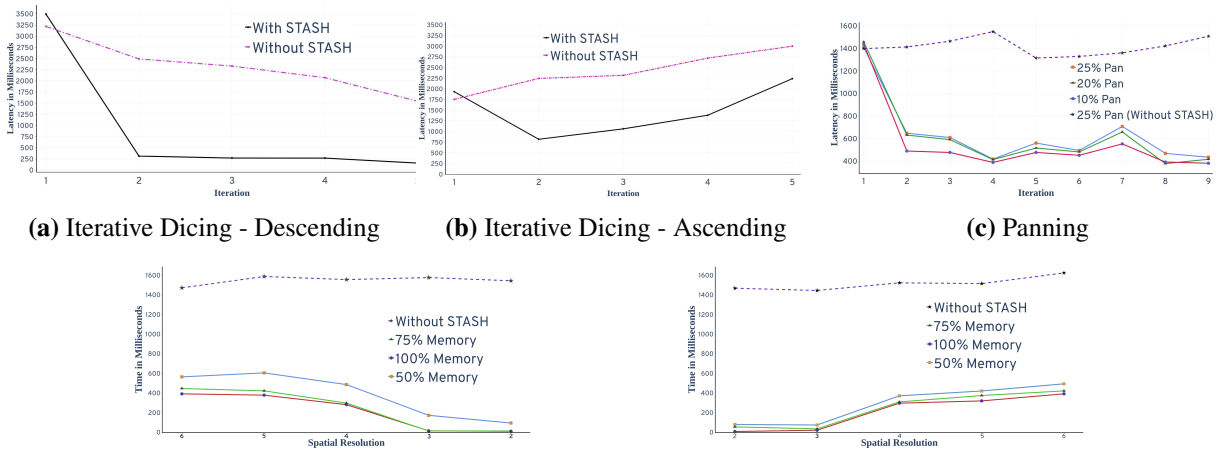
(d) Autoscaling

Figure (5.5) Performance Evaluation of STASH: (a) and (b) show effects of query size on its latency and throughput, respectively; (c) compares STASH maintenance time for different query sizes and (d) shows the improvement in throughput for STASH's replication mechanism over normal execution during hotspot.

Fig.5.5a shows that STASH with all necessary Cells in-memory outperforms the other two scenarios with ∼5x improvement over no STASH scenarios for large query sizes such as country and state. Hence, a fully populated STASH graph helps transform even large queries to interactive operations.

STASH Maintenance Overhead

In Fig.5.5a, the average latency in the worst case scenario is slightly more than in the no STASH case, which can be explained by the overhead in the unsuccessful look-up for matching Cells in the graph and then looking into the disk. The population of the Cells fetched from disk to memory is done at the back-end in a separate thread. Fig.5.5c shows the cold-start scenario where all the Cells from a query have to be inserted in-memory and the time taken population that goes down considerably with query size, since lesser Cells are to be inserted in STASH.



(a) Iterative Dicing - Descending **(b)** Iterative Dicing - Ascending **(c)** Panning

(d) Drill-Down **(e)** Roll-Up

Figure (5.6) Query Performance Evaulation for Different Common Visual Analytics Operations

5.3.3 Visual Exploration With Collaborative Caching

In practical scenarios, fragments of the query's spatiotemporal extent will be contained in STASH graph while the remainder needs to be fetched from disk as follows:

Iterative Dicing

To simulate the user action of sequentially increasing and decreasing the query area, we have implemented ascending and descending iterative dicing respectively, as shown in Fig.5.6b and Fig.5.6a respectively. It shows a sequence of 5 queries that, keeping the spatiotemporal resolution fixed, varies the Query_Polygon size in either ascending order or descending order. We can see

that descending iterative dicing performs much better for a STASH enabled system since a larger area (country level) is fetched in the first query and then, iteratively, a subset of the first query (20% spatial area reduction) gets queried (final query having size $\sim$(5.2°,10.4°)) - leading to all necessary Cells existing in memory from the second query onwards. The ascending version is the previous set of queries executed in reverse order. Here, as the spatial extent increases, a fraction of the relevant Cells are found in-memory, which does lead to improved performance over the basic system, but not to the extent of the descending version.

Zooming

We replicate the scenario of a user sequentially increasing or decreasing the resolution of a view area by two sets of experiments - drill-down (zoom-in), where a user starts with a lower spatial resolution of 2 of a state-level area and then recursively increases the resolution to 6 that incurs $\sim$32 fold increase in the number of possible Cells at each step. Roll-up (zoom-out) is the reverse of the drill-down operation. To compare the performance of our system in scenarios with varying amount of relevant cells in-memory, we have randomly stacked the STASH graph with regions covering 50%, 75% and 100% of all the relevant Cells.

Fig.5.6d and Fig.5.6e contrasts the latency of the drill-down and roll-up scenarios, respectively, for a STASH enabled system against the basic system. As expected, more the amount of relevant Cells in-memory, the better the latency. However, in all scenarios with partial information, we see at least 40% improvement in latency over a system without STASH.

Panning

We replicate panning in our experiments by starting with a state-level query and moving the rectangle by a certain amount(10%, 20%, 25%) in 8 possible directions around the starting rectangle. So, the first query encounters an empty STASH graph and then, from the second query onwards, a fraction of the necessary Cells should exist in-memory. The results in Fig.5.6c support our assumption. We see that the basic analytics system has uniformly high latency, whereas, that in STASH enabled system is considerably low. The lower the amount of pan, the larger is the over-

lapping area between two consecutive queries, which would benefit a STASH enabled system, as validated by Fig.5.6c. Also, the comparison of 25% pan scenario between a basic and a STASH enabled system shows considerable improvement - ranging from $\sim$ 73%-60% reduction in latency.

Throughput

Fig.5.5b shows the throughput of a STASH-enabled system vs that of a basic system. This experiment involves firing 10,000 county-level requests over the cluster which are created by selecting 100 random rectangles (of sizes state, county and city) over the globe and then randomly panning around each by 10% in any random direction 100 times, to replicate spatiotemporal locality of requests. The throughput is calculated based on the total time taken for the last request to be executed successfully. A STASH-enabled system shows $\sim$5.7x, $\sim$4x and $\sim$3.7x improvement in throughput for state, county and city-level queries, respectively.

5.3.4 Improvement Through Autoscaling

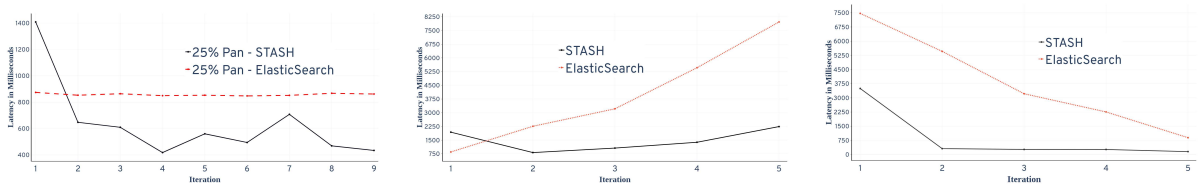
Fig.5.5d contrasts the performance of STASH's replication scheme against STASH without dynamic replication under skewed traffic. We simultaneously executed 1000 county-level requests, by randomly panning around a random starting point, to emulate the hotspot scenario of sudden interest over a single region from multiple users. Our system was configured to initiate Clique handoff with pending requests of over 100. To compare improvement caused by a replication operation, the cooldown time was set high. Fig.5.5d shows the number of responses received each second from the start. We can see that STASH with a dynamic replication scheme processes larger number of queries per second and finishes all tasks $\sim$ 20 seconds before STASH without dynamic replication.

5.3.5 Comparison with ElasticSearch

We contrasted STASH's performance against ElasticSearch, which has its own caching system, with some of the previously OLAP scenarios with consecutive overlapping requests.

Panning

The panning scenario when replicated on ElasticSearch gives results as shown in Fig.5.7a. We can see that STASH shows better improvement in performance, whereas ElasticSearch's latency improves slightly. At each step with the latency-reduction with respect to the latency of the first request with STASH ranges between $\sim$ 70% and 49.7%, whereas that of ElasticSearch stays between $\sim$ 2% and 0.6%. Also, the second query onwards, STASH's latency is significantly lower which demonstrates better management of in-memory data in case of overlapping queries.



(a) Panning **(b)** Iterative Dicing - Ascending **(c)** Iterative Dicing - Descending

Figure (5.7) Contrasting STASH's Latency Against ElasticSearch for Common Visual Analytics Operations

Dicing

Fig.5.7b and Fig.5.7c compares the results of the ascending and descending iterative dicing experiments, as mentioned above, between STASH and ElasticSearch. Here also, we see that STASH shows a much steeper drop in latency from the second query onwards by effciently utilizing the common Cells stored in-memory for the subsequent queries.

Chapter 6

Compressed/Low-dimensional Representation of Data

Spatiotemporal queries over voluminous datasets with high data density are compute-intensive and require processing of a huge amount of records for extraction of information such as aggregate statistics or patterns over a given sub-domain. Voluminous datasets with high-dimensional data collections, like in the case of remotely-sensed hyper-spectral satellite imagery, offer a wide scope to identify phenomena or underlying patterns and inform decision-making. Apart from their volumes, the nature of the data objects contained in these collections introduces challenges stemming from their complexity and spatiotemporal resolutions.

In such cases, compressed data representations reduce the storage footprint of these datasets, thereby allowing a more significant portion of the data to be cached within limited memory resources. These compressed representations are used in place of actual data (in the STASH graph) for query analytics. This, in turn, mitigates the need for frequent disk I/O operations, resulting in improved query performance and overall system efficiency. We explore 3 different strategies for handling these kinds of scenarios:

1. **Hierarchical Data Cubes**

2. **Low-resolution Data Representations for Model-Driven Reconstruction**

3. **Extracting low-dimensional representation (embeddings)**

The efficacy of each of these modes of compressing raw data depends on the specific problem requirements. The choice of which strategy to use depends on the nature of the underlying dataset and is influenced by the degree of compression desired as well as the amount of accuracy or perceptual quality of the visualization results that is desired.

6.1 Hierarchical Data Cubes (RUBIKS)

Data cubes [94] are considered an effective mechanism to facilitate such summarizations [40, 94, 95]: we extend this concept of data cubes to spatiotemporal data spaces where the number of observations may be very large. Crucially, we support these aggregations at scale, with low latency, alongside the ability to perform these operations along diverse spatial hierarchies (administrative, watersheds, quadtiles, etc.). We explore these ideas in the context of our research prototype, RUBIKS [96].

We leverage a novel mix of algorithmic, statistical and systems approaches to facilitate real time data summarization at scale across user-specified spatiotemporal scopes. Our methodology places no constraints on the size of these spatiotemporal scopes. Data from moderately sized spatial extents (e.g., tracts, counties, or watershed boundaries) are collated and stored in-memory for faster evaluations at runtime.

Summarizations allow a researcher to spot patterns that arise at diverse spatiotemporal scopes. Summarizations provided by our data cubes include min, max, mean, median, variance, standard deviations, and distributional skew and kurtosis associated with individual features (or variables). We also supplement these measures by tracking the covariance across a set of user-specified features. RUBIKS data cubes support pivot, aggregation, and disaggregation operations. Pivots allow the data cube to be probed across a specific dimension e.g., spatial, temporal, or any of the features encapsulated within the cube. The roll-up and drilldown operations relate to aggregation and disaggregation operations across spatiotemporal scopes. For example, a user may be interested in exploring the data space at coarse scales (roll-up) or at finer scales (drilldowns). These operations allow a user to specify interest over the dataspace at progressively larger or smaller spatial extents, time ranges, or combinations thereof.

Rather than compute these data cubes exhaustively every time a query is issued, we perform a limited number of one-time precomputations that we then leverage to support data cube operations. The smallest unit of data summarization in the system is a ***cubelet*** representing the smallest, indivisible spatiotemporal scope at which summarizations are performed. In our methodology, the

scope associated with the cubelet is configurable. Data cubes are constructed from cubelets. Data cubes may either be constructed from cublets or hierarchically constructed from other cubes. We leverage Welford's algorithm to compute the cubelets in an online, single-pass fashion, and also acount for irregular timeseries observations. Information maintained within the data cubes are also amenable to leveraging the same online method to compute data cubes at ever coarser scales. Our methodology also allows data cubes to be constructed from non-contiguous spatiotemporal cubes.

In RUBIKS, cubelets are space-efficient and can be persistently stored in since they are used in the computation of data cubes that may span diverse spatiotemporal scopes. Persistent storage of the cubelets also precludes duplicate computations alongside repeated sweeps of the data involving I/O. The data cubes, on the other hand, are ephemeral meaning they are garbage collected after a period of time.

The STASH cache is used to store data cubes that have been calculated based on user-specified queries. We also store cubelets that were used to construct these data cubes; the rationale for this is that it is often the case that users are incrementally refining queries to customize the spatiotemporal scopes of interest. As such, cubelets that are part of a query have a higher likelihood of inclusion in the refinement queries. Second, the cache can reduce duplicate processing alongside any I/O that such refinements entail.

RUBIKS also supplements the summarization feature with the ability to visualize these summarizations using a Choropleth map to render spatial variations of measures of interest. As such, the queries may be composed visually, and the roll-ups and drilldowns can be performed using slider bars. We allow dynamically constructed data cubes to be visualized using our Choropleth map service.

6.2 Low Resolution Data Representations (GLANCE)

This strategy focuses on populating the in-memory graph with low-resolution representations of large data objects and reconstructing them through super-resolution at runtime. For interactive visual analytics applications, data retrieval and processing are driven by users' actions. Drilling

Figure (6.1) The three rows demonstrate front-end visualization results from 2x, 4x and 8x upscaling scenarios, respectively. The columns represent the model inputs, the SR outputs from the basic and partial SR models and the ground-truth high-resolution image, respectively.

down of the current view allows isolation of finer-grained details, while telescoping of a view allows users to contrast surrounding values. Frequent and flexible visual analytics operations result in a very large number of data retrievals from the remote data storage. Though it might be possible for certain user operations to be performed locally (e.g. aggregations for zoom-out) with data stored in a local cache, enhancing resolution (e.g. zoom-in) of the current view cannot avoid accesses to the backend storage service.

The GLANCE framework delivers images at diverse resolutions, through *in-situ* reconstruction, while significantly reducing data communications between the user and the backend storage system. As shown in Fig.6.1, GLANCE constructs a model that captures non-linear relationship between images with different resolutions to generate super-resolution (SR) image from lower resolution images. Our models, *GlanceNet*, comprises two Generative Adversarial Networks (GANs) to address the basic upscaling scenario along with a partial upscaling scenario.

The GlanceNet models are a variation of *progressive GAN* that infers higher resolution images from aggregated (low-resolution) images. The primary benefit of using a progressive GAN is generating a consolidated model that captures interactions that exist between arbitrary pairs of levels across the entire spectrum of resolutions. For instance, in the case of satellite observations, the details and interactions for drill-down operations from a city-to-block are different from those needed for a drill-down from a metropolitan area to the city. The enhancement GAN targets the case that the high resolution image is partially available in the memory.

GLANCE also provides a refinement to maintain sufficient accuracy of inferred images while balancing data transfer from the backend storage server. The models may output images with unsatisfactory accuracy due to an insufficient number of input tiles or unexpected image complexity due to spatial variations. For example, the model may work well for three successive drill-down operations for regions in Nevada but may meet our accuracy thresholds only for two successive drill-downs for regions in Florida. To detect and avoid potential error-prone images, the system tracks the regional model accuracy and identifies users' requests that will benefit from data retrievals from backend servers.

6.3 Low-dimensional Representations using Encoder-Decoder Network (ARGUS)

This aspect of our framework deals with in-memory computing over low-dimensional *sketches* of high-dimensional data objects, guided by model-assisted insight extraction. We attempt to create a versatile low-dimensional latent representation out of underlying data objects (eg. multi-spectral satellite imagery) that can help - 1) reduce memory footprint of the in-memory data objects and 2) facilitate faster data processing and reduce response time. Next we introduce the different components of this framework (see Fig. 6.3). In order to extract intuitive and meaningful insights from such data objects and maintain low latency for our queries, we implement our model-driven extraction of insights.

The first part of this framework for rapid processing of high-dimensional data is a trained encoder-decoder network which can be used during data ingestion to reduce the dimensionality of the incoming multi-spectral data, while sufficiently preserving its contents. The network is aimed at generating embeddings that represent a higher-order latent representation of the actual data object. We plan on implementing convolutional autoencoders for this purpose since they have shown promising results over satellite image data [97], especially in feature extraction.

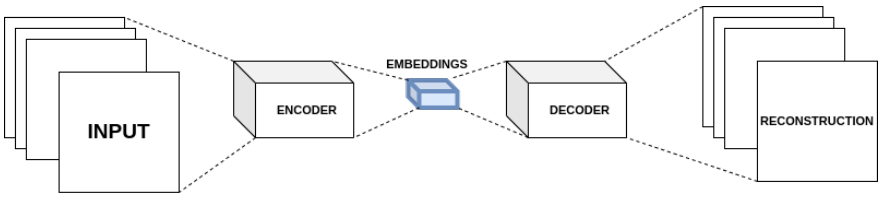


Figure (6.2) Typical Autoencoder Network

Autoencoders [98] based on deep neural networks are commonly used for a number of different applications, including feature extraction and dimensionality reduction. The driving principle behind an autoencoder is that the high dimensional data has a significantly smaller lower-dimensional embedding in a latent space that is sufficient to represent the content of the original data. Autoencoders are an unsupervised model that work by reducing a given input down to a small vector (the embedding), then optimizing the network to reconstruct the input or a version of the input, from that embedding.

A simple Autoencoder network consists of 3 parts (Fig. 6.2) - an *encoder* made from a sequence of convolutional layers, which help reduce the size of the input through a set of filters, the generated latent embeddings and a *decoder* that reconstructs the input from the latent embeddings. The decoder layers are a mirror image of the encoder and are trained to reconstruct the input from the embedding generated from the encoder. These two modules together help optimize the quality of the embeddings generated. A convolutional autoencoder is a modified version of a simple autoencoder, where the encoder and decoder layers, instead of being composed of fully-connected layers now consist of a sequence of convolution layers (and pooling layers). The structure of the encoder and decoder still mirror each other.

6.3.1 Supervised Model Building Using Embeddings

The embeddings generated from the previous component are expected to have significantly low memory-footprint compared to the original data-object. This helps improve the number of elements in our cache and increases the hit-rate. However, performing post-processing on these embeddings requires us to build models capable of extracting insights from such latent representations. Additionally, making these embeddings need to be versatile enough to serve multiple types of analytics/ geoprocessing introduces a new set of challenges. To ensure that the training of these models needs to be collaborative.

We use this second set of models during actual query-evaluations that involve post-processing (for example, calculating slope from an elevation raster file) using in-memory of on-disk embeddings of the actual data. While extracting insights from embeddings, instead of actual input, can be challenging, one advantage to training these modules out of embeddings is that it enables us to train over much smaller inputs which would make training faster. This also could potentially help us build models that are lightweight. As a result, they should also have faster convergence rates.

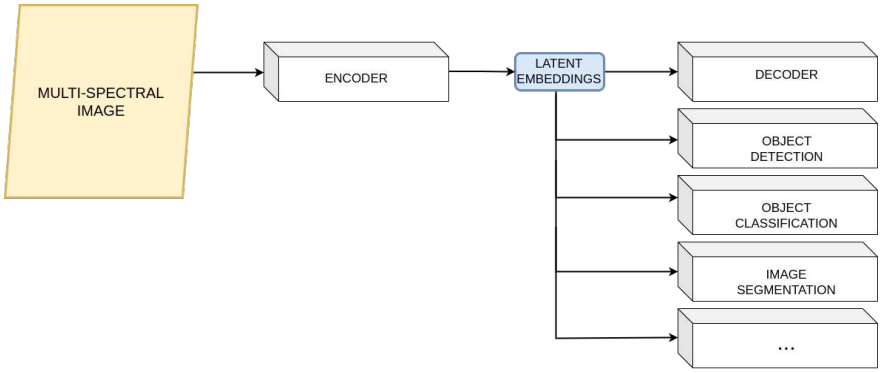


Figure (6.3) Overview of Insight-Extraction Framework

6.3.2 Multi-task Learning

Since our embeddings need to be multi-purpose and serve extraction of different types of analytics, their training needs to support a mutual exchange of knowledge. This would enable our embeddings to be truly versatile and also help in its robustness by assimilating knowledge across

multiple domains. In this context, *Multi-task learning* (MTL) [78] is an effective solution (Fig. 6.3), where we aim to train multiple models that learn related tasks jointly to leverage exchange of knowledge that facilitates generalization of the individual models. Feature-based multi-task learning aims at learning common features among related tasks as a way to exchange common knowledge. Multi-task learning involves training of machine learning models with data from multiple tasks simultaneously, using shared representations. This enables the models to acquire shared knowledge between a set of related tasks. These shared representations increase data efficiency and can potentially yield faster learning speed for related or downstream tasks, helping to alleviate the well-known weaknesses of deep learning: large-scale data requirements and computational demand.

Next, we individually discuss each of these compression strategies and the approach of reconstructing/ generating aggregate statistics from them. We also discuss their improvement in terms of interactivity as well as the accuracy of their results.

Chapter 7

Hierarchical Data Cubes (RUBIKS)

RUBIKS facilitates construction of hierarchical datacubes which may be constructed from cubelets (smallest, indivisible unit of aggregation in the system) or from other data cubes. Data cubes generate aggregated summarization of measurements over a spatiotemporal scope at varying levels of coarseness, based on their resolution. Here, we demonstrate our methodology for computing and analyzing data cubes over disjoint spatiotemporal extents.

7.1 Query Evaluation

The following is a sample spatiotemporal query that we support at client-side. RUBIKS supports spatiotemporal queries at varying levels of resolution (spatial_resolution, temporal_resolution) over any given viewport (Polygon) and timerange(Query_Time).

```
select pearson(iron,mercury), ...
from Aqua_Dataset
where coornidates in Polygon
and time_stamp in Query_Time
group by spatial_resolution, temporal_resolution
```

Fig. 7.1b provides an insight into the query evaluation process orchestrated by RUBIKS. The system handles analytical queries over spatiotemporal data through the utilization of datacubes. We elaborate on the overall query evaluation mechanism of RUBIKS in a distributed context. When a client query is initiated, it is initially directed towards the relevant cluster nodes (as explained further in the subsequent section). At each node, a search is conducted within the in-memory cache for cubes that either precisely match or can be repurposed to meet the requirements of the current query. Subsequently, the query is enhanced to retrieve any unfulfilled spatiotemporal extents from the backend storage.

In the backend storage, RUBIKS engages in a search for ephemeral cubes that can be effectively employed or repurposed to furnish accurate responses for the ongoing query. For any cubes that are found missing, they are dynamically constructed from the collection of perennial cubelets and subsequently dispatched to the requesting node. In addition to addressing the query at hand, these newly formed data cubes are added to the roster of ephemeral cubes for future potential use. Data cubes created using this dynamic and targeted process are cached. By orchestrating this interplay of cache utilization, dynamic data cube construction, and query enhancements, RUBIKS realizes a robust and responsive query evaluation mechanism that ensures efficient utilization of available data and computational resources. Crucially, correctness is preserved while ensuring efficient analysis of spatiotemporal datasets.

7.2 Cubelets

In the RUBIKS framework, a cubelet serves as the fundamental unit of aggregation and analysis. These cubelets play a pivotal role in encapsulating aggregated values across a diverse spectrum of measurements within a well-defined spatiotemporal scope. With these Cubelets, we develop a dynamic analytical framework that facilitates iteratively identifying regions of interest that satisfy desired properties or covariences.

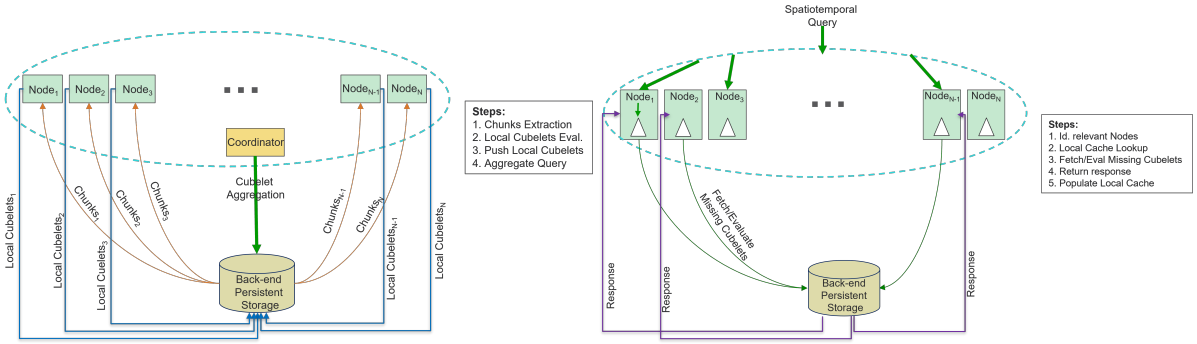
Our cubelets encapsulate statistical summaries such as counts, means, minimums, maximums, and standard deviations, alongside distributional skew and kurtosis, for all observations for a particular variable within the specified spatiotemporal extent. The data encapsulated within a cubelet is space-efficient and is amenable to aggregations i.e., cubelets can combined to produce a new data cube that encapsulates the aggregated measures of interest.

The ability to hierarchically aggregate cubelets (and cubes) facilitate a comprehensive exploration of spatiotemporal patterns, trends, and relationships across the entirety of the dataset's geographic and temporal domain. By orchestrating queries that target data cubes, *we aim to pinpoint regions of interest that align with specific criteria or exhibit correlated behaviors with a predefined*

set of features, enabling targeted analysis of intricate spatial and temporal phenomena and extraction of nuanced insights, contributing to informed decision-making in a wide array of applications.

Cubelets are created over a configurable spatiotemporal scope. The cubelets can be aggregated hierarchically into data cubes at varying spatiotemporal resolutions. We describe the hierarchical organization of cubelets in section 7.7.

In RUBIKS, cubelets are *perennial* while the data cubes are ephemeral. Cubelets are created and persisted on stable storage (and thus are perennial) along with actual data points during ingestion. These cubelets have a predetermined resolution and constitute the lowest level of the cube hierarchy. Cubes are coarser in the sense that they are computed on an on-demand basis from cubelets or other cubes in a hierarchical fashion and are ephemeral (i.e., they may or may not be persisted to disk).



(a) Cubelet Construction During Ingestion

(b) Cubelet Fetching/ Dynamic Evaluation During Queries

Figure (7.1) RUBIKS Cubelet Contruction and Fetching

7.3 Cubelet Content

Cubelets summarize data from a particular spatial extent and are constructed from persistent data stored on disk (we place no constraints on the storage framework used to store such data). Each cubelet summarizes data for a configurable but system-wide spatiotemporal scope. The cubelet comprises a set of metadata attributes recorded within that region. The supported metadata

includes essential statistical measures such as count, mean, minimum, maximum, and standard deviation for each attribute.

To enhance the analytical capabilities of cubelets, for a predefined set of attribute pairs, we also maintain running covariances within each cubelet. These covariances facilitate the evaluation of Pearson correlation coefficients at runtime, enabling researchers to gain insights into the relationships between different attributes within the cubelet.

7.4 Cubelet Spatiotemporal Bound

RUBIKS offers the flexibility to construct data cubes at different spatiotemporal extents, tailored to the specific dataset, creating non-overlapping regions as the foundation for construction of data cubes. In RUBIKS, we allow data cubes to be created over varying types of disjoint geospatial bounds, such as quadtiles, Hydrologic Unit Codes (HUC) [99], and Federal Information Processing Standards (FIPS) codes that are used by the U.S. Census Bureau. This feature enables the analysis of data with diverse spatial characteristics, accommodating datasets that might have irregular or complex geographical boundaries. By supporting multiple geospatial bounds, RUBIKS allows researchers to perform detailed analyses on localized regions while also gaining insights into broader geographic trends, fostering a more comprehensive exploration of spatiotemporal patterns and relationships within the data.

Perennial cubelets represent the finest level of aggregation and are persisted both on-disk over our distributed storage, as well as in-memory cache that we construct over the cluster nodes. These can be hierarchically combined to create coarser aggregates – the ephemeral cubes – facilitating a multi-resolution analysis of spatiotemporal patterns and trends. This provides a powerful tool for efficient and flexible exploration of large-scale point datasets with varying granularities. Ephemeral cubes are constructed as client-queries get evaluated server-side to enable collaborative query evaluation. At the finest level, perennial cubelets are constructed and updated during data ingestion by aggregating and summarizing point data that fall within a predefined spatiotemporal

extent – for instance, over a spatial bound of a HUC12 boundary and temporal bound of a single day.

7.5 Distributed Ingestion: Perennial Cubelet Generation

Perennial Cubelets are generated during data ingestion. In Fig. 7.1a, we illustrate the process of generating these cubelets. Preprocessing of incoming voluminous data in a standalone fashion can be time-consuming and compute-intensive. Rubiks relies on a distributed cluster of nodes for handling data ingestion, cubelet creation and query evaluation.

During ingestion, incoming data-points are partitioned into chunks and ingested in a distributed manner across our cluster nodes. Each node independently computes its local set of cubelets, contributing updates to a temporary set of cubelets in the distributed storage backend. Subsequently, a coordinator node initiates an aggregation query to combine local cubelets with overlapping keys, if any, into usable perennial cubelets. Only cubelets are constructed and persisted; data cubes themselves are constructed hierarchically on an on-demand basis.

7.6 Cubelet Update

To adapt to the continuous updates to the underlying storage, we ensure concurrent data ingestion and the update of cubelets in persistent memory.

7.6.1 Welford's Algorithm for Rapid construction/Updates

To ensure efficiency and scalability within RUBIKS, we employ dynamic merging and updates of cubelets using Welford's algorithm, which provides a computationally efficient (single-pass) approach for incrementally calculating the mean and variance as new data are added or cubelets are merged. This method allows for real-time updates and analysis without the need to recompute the entire dataset, reducing both computational complexity and memory requirements.

Leveraging Welford's algorithm and associated metadata mean that our cubelets can efficiently accommodate data updates and adapt to changing input without sacrificing analytical accuracy.

The algorithm's incremental, online nature makes it particularly well-suited for handling continuous data ingestion and maintaining up-to-date statistics within cubelets and across data cubes that are hierarchically constructed using cubelets and other data cubes. As a result, both cubelets and data cubes can dynamically adjust to new data points, supporting real-time analyses and ensuring a robust and scalable solution for data management and analysis. Utilizing Welford statistics for aggregation over cubelets allows us to 1) rapidly identify cubelets that require change/creation, and 2) perform rapid, decentralized updates over our cluster.

Due to the disjoint nature of measurements of attributes at a monitoring station, recorded measurements of any pair of desired attributes at the same location is never guaranteed to be concurrent. This complicates the process of measuring correlation between such attributes. To account for this situation, we aim to estimate the correlation measures in these cases by interpolating the recorded values based on how distant their measurements are in time. We explain the process of correlation computation for such misaligned measurements next.

7.6.2 Correlation estimation for misaligned time series

If two time series x and y are observed at irregular time points $\{s_i\}_{i=1}^{n_x} \neq \{t_j\}_{j=1}^{n_y}$, the empirical means $\widehat{\mu}_x, \widehat{\mu}_y$ and empirical standard deviations $\widehat{\sigma}_x, \widehat{\sigma}_y$ for the two series can be computed directly. The empirical correlation, however, can only be computed directly if the observation times are aligned ($n = n_x = n_y$, $s_1 = t_1, s_2 = t_2, \ldots, s_n = t_n$):

$$\widehat{\rho}_{xy} = \frac{1}{n-1} \sum_{j=1}^{n} \left\{ \frac{x(t_j) - \widehat{\mu}_x}{\widehat{\sigma}_x} \right\} \left\{ \frac{y(t_j) - \widehat{\mu}_y}{\widehat{\sigma}_y} \right\}.$$

If observation times for the two series are misaligned, we use the non-rectangular kernel approach described in [100] to approximate the correlation. Let

$$K_h(s,t) = \frac{1}{h\sqrt{2\pi}} \exp\left\{ \frac{-(s-t)^2}{2h^2} \right\}$$

denote the Gaussian kernel function with bandwidth parameter h. This kernel function is used to determine which time points between the x and y series are close enough to be used in estimating the correlation, via

$$\begin{aligned}\widetilde{\rho}_{xy} &= \frac{1}{\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} K_h(s_i - t_j)} \\ &\times \left[\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} \frac{x(s_i)}{\widehat{\sigma}_x}\frac{y(t_j)}{\widehat{\sigma}_y} K_h(s_i - t_j) \right. \\ &- \frac{\widehat{\mu}_y}{\widehat{\sigma}_y}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} \frac{x(s_i)}{\widehat{\sigma}_x} K_h(s_i - t_j) \\ &- \frac{\widehat{\mu}_x}{\widehat{\sigma}_x}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} \frac{y(t_j)}{\widehat{\sigma}_y} K_h(s_i - t_j) \\ &\left. + \frac{\widehat{\mu}_x}{\widehat{\sigma}_x}\frac{\widehat{\mu}_y}{\widehat{\sigma}_y}\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} K_h(s_i - t_j) \right]. \end{aligned} \tag{7.1}$$

We compute the average distances Δ_s, Δ_t between consecutive time points in $\{s_i\}_{i=1}^{n_x}$, $\{t_j\}_{j=1}^{n_y}$, respectively, and choose $h = 0.25 \times \max\{\Delta_s, \Delta_t\}$, following [100]. As noted in [100], $\widetilde{\rho}_{xy}$ is not guaranteed to lie within $[-1, 1]$; we set it equal to the closest boundary value if it falls outside.

If information from two cubelets is to be combined, let $\{s_i^{(k)}\}_{i=1}^{n_x^{(k)}}$ and $\{t_j^{(k)}\}_{j=1}^{n_y^{(k)}}$ denote the observation time points for cubelets $k = 1, 2$. Assume that from pilot analysis a single value of h can be determined across cubelets. Further, assume that

$$K_h\left(s_i^{(1)}, t_j^{(2)}\right) \simeq 0, \quad K_h\left(s_i^{(2)}, t_j^{(1)}\right) \simeq 0;$$

that is, a misaligned pair in two different cubelets has time points sufficiently far apart to contribute nothing to the correlation computation. Then replace each cubelet mean and standard deviation in equation (7.1) by the combined mean and standard deviation; and replace each double sum in (7.1) by adding the two corresponding double sums (one for each cubelet); e.g., replace the first double

sum in the numerator by

$$\sum_{k=1}^{2}\sum_{i=1}^{n_x^{(k)}}\sum_{j=1}^{n_y^{(k)}}\frac{x(s_i^{(k)})}{\widehat{\sigma}_x}\frac{y(t_j^{(k)})}{\widehat{\sigma}_y}K_h(s_i^{(k)}-t_j^{(k)}).$$

In addition to the information already required for updating the mean and standard deviation when combining cubelets, this correlation computation requires storing for each cubelet the four distinct double sums in (7.1).

7.6.3 HashGrid for Updating Cubelets

In RUBIKS, the need for continuous cubelet updates to ensure query accuracy stems from the dynamic nature of the underlying data store. To effectively accommodate this evolving data landscape, concurrent data ingestion and cubelet updates within persistent memory are pivotal. This process is orchestrated through a hashgrid-driven approach, aimed at ensuring accuracy of constructed data cubes with the evolving dataset through the following steps:

Binary Hierarchical Hashgrid: RUBIKS maintains a binary hierarchical hashgrid, wherein each element corresponds to a specific cube. This hashgrid serves as a reference to indicate whether a cube is up-to-date or requires updating due to changes in the underlying data.

Coordinator-Initiated Updates: During execution of the aggregation, the coordinator node also monitors and tracks cubes that have undergone modifications since the last update. The coordinator node updates the hashgrid based on the modifications detected. Each corresponding element in the hashgrid is updated to reflect the current status of its respective cube – indicating whether it is up-to-date or not.

Hierarchical Update Propagation: The hierarchical structure of the hashgrid streamlines the propagation of updates. The coordinator node can efficiently update higher-level hashgrid elements based on changes in the lower levels. This hierarchical mechanism ensures a streamlined and efficient update process.

Cluster-Wide Synchronization: Once the hashgrid is updated by the coordinator, this updated

Table (7.1) Cubelet Generation: Comparison between time (seconds) taken to create Cubelets in a cold-start scenario vs daily updates

	County	**Quadtiles**	**HUC-12**
Cold-Start	187.53	219.90	363.77
Daily Updates	4.91	5.84	17.74

hashgrid is disseminated to all the cluster nodes. This push informs each node about the cube that are currently out-of-sync and cannot be used for query evaluation due to outdated information.

By leveraging this hashgrid-driven approach, RUBIKS seamlessly incorporates continuous data updates into its cubes. This process ensures that the cubes remain relevant and accurate, enabling accurate and up-to-date query evaluations even over dynamic, continually-evolving datasets.

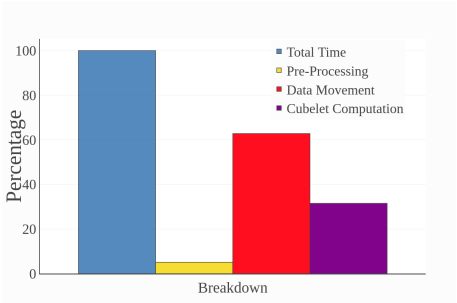


Figure (7.2) Breakdown of Overall Cubelet construction time

We evaluate the time taken to construct the *perennial cubelets* over RUBIKS in a cold-start scenario, where we have to ingest $\sim 226M$ entries into our distrubuted storage. Table 7.1 compares the time taken to construct cubelets constructed over varying non-overlapping geospatial bounds. We can see compared to the total number of records being ingested, the overall time to construct cubelets is quite low, in the order of a few minutes. Additionally, we note that the overall time taken to construct the cubelets is directly proportional to the total number of cubelets being constructed. For instance, for the total geospatial extent of the CONUS, the number of unique counties is 3163, the total number of quadtiles within the bounds is $\sim$ 15,000, whereas the total number of HUC-12 regions is $\sim$87,000, which directly impacts the total number of cubelets required to be created and

saved into our framework. As expected, the overall cubelet construction time is proof of that and it is to be noted that ideally, cubelet construction would occur alongside data ingestion.

We can also see that the update time for cubelets for incoming daily measurements is significantly low compared to a cold-start scenario, as expected. The difference in times taken for various geospatial cubelet bounds is also reflected here, as in the case of the cold-start scenario. We also profile a break-down of the overall operation of cubelet construction during ingestion. During a cold-start generation of perennial cubelets, we need to fetch the relevant distributed data to each computing node, perform pre-processing to load shapefiles to identify specific geospatial cubelet boundaries and perform aggregation and computation of cubelets, followed by persisting them. Fig. 7.2 demonstrates the percentage of overall cubelet generation time for a cold-start scenario where we create county-wise cubelets for each month over records starting from 1970 till current day. We note that data movement, which constitutes both moving ingested data to the cluster nodes and computed cubelets back to the persistent storage takes up a majority of the overall time.

7.7 Hierarchical Aggregation of Cubelets

The computed cubelets, which represent fine-grained spatiotemporal aggregates, are systematically organized into a hierarchical structure. This hierarchical organization is achieved through the aggregation of lower-level cubelets that lie within the bounds of a given *parent* cubelet, ensuring efficient representation and management of the cubelets. Additionally, specific hierarchical structures such as quadtiles, Hydrologic Unit Codes (HUC) and Federal Information Processing Standards (FIPS) codes are employed to cater to diverse geospatial bounds. Temporally, we allow aggregation to be in units of days, weeks, months, or years.

To form coarser aggregates at higher levels of the hierarchy, cubelets are combined. This merging process allows the creation of larger aggregations, providing a multi-resolution perspective of the data. Moreover, higher-level spatiotemporal extents are applied to encompass multiple cubes of varying types, further enhancing the versatility of the hierarchical framework. By employing these

methods, the hierarchical organization enables more insightful analysis of spatiotemporal patterns and trends within the datacubes.

The cube hierarchy (with cubelets at the lowest level and dynamically, recursively constructed data cubes) is maintained in the form of a metadata graph. However, since we can deterministically and hierarchically aggregate based on spatiotemporal bounds, there is no need to maintain actual links between the cubes themselves. We maintain these cubes as a set of hashmaps, grouped by their spatial and temporal keys, allowing targeted, efficient $O(1)$ retrievals.

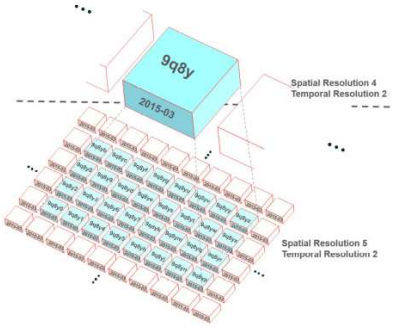
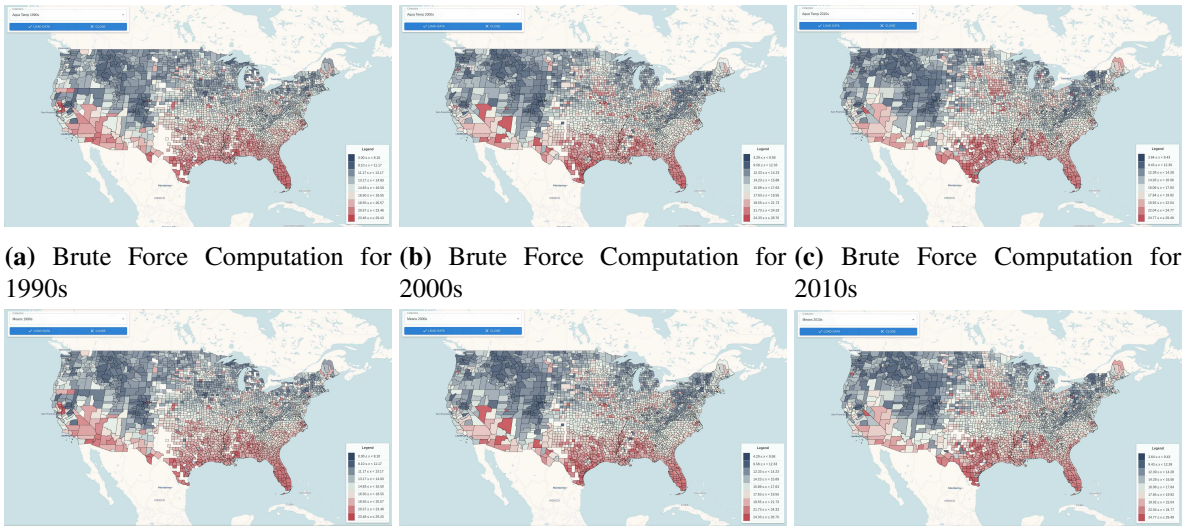


Figure (7.3) Cubelet Spatiotemporal Bounds

7.8 Visualization of Cubelets

Cubelets allow Exploratory Data Analysis over backend data such as heatmaps, time series plots, and interactive visualizations to identify patterns and trends across different levels of the Cubelet hierarchy. We have used cubelets to generate heatmaps of water contaminant proliferation at the county, watershed boundary, and individual water body level across the continental United States. We used a JavaScript front-end leveraging the DeckGL mapping framework to visualize heatmaps.

Since the RUBIKS cubelets form the backbone of its query evaluations, we compare the accuracy of RUBIKS' aggregate statistics against those computed through brute force. Since we use Welford's online algorithm to compute and update our cubelets, we expect them to accurately rep-



(a) Brute Force Computation for 1990s **(b)** Brute Force Computation for 2000s **(c)** Brute Force Computation for 2010s

(d) Cubelet-based Computation for 1990s **(e)** Cubelet-based Computation for 2000s **(f)** Cubelet-based Computation for 2010s

Figure (7.4) Comparison of Accuracy of Summary Statistics

resent independent statistical measures such as mean, standard deviations, skewness and kurtosis, which are derived from the first four order of moments.

Fig. 7.4 shows choropleth maps computed using both brute-force and through aggregation of RUBIKS' perennial cubelets. The geospatial bounds used here are US counties. We compare the mean for the decades 1990s, 2000s, and 2010s for the water temperature in Celsius. We can see that along with significantly improved fetch latency (as shown later), we get high accuracy with our cubelets constructed in an online fashion.

We profile the improvement in latency through our RUBIKS framework, compared to that of a spatiotemporal query over raw data. Here, we profile the latency over queries of varying size. We evaluate the time taken to compute county-wise aggregate statistics per month. By keeping the spatial bounds of the query fixed to the entire CONUS, we vary the temporal extent of the query to a month, a year and a decade. Table 7.2 profiles the average time taken for each of these 3 types of queries with and without the use of RUBIKS cubelets. We can see significant improvement in query times compared to fetching of raw data, with improvement ranging from ∼3800-2000x.

Table (7.2) Spatiotemporal Query: Comparison between latency (seconds) for varying sizes

	1 Month	**1 Year**	**10 Years**
RUBIKS	3.3	3.32	6.51
Brute Force	12556.6	12606.2	12686.9

Chapter 8

Model-Driven Data Reconstruction (GLANCE)

Frequent and flexible visual analytics operations result in a very large number of data retrievals from the remote data storage. To further enable interactive visual analytics over high-dimensional data at scale, we complement the STASH framework with a set of data reconstruction modules. These additional modules enable model-driven rendering of multi-resolution phenomena leveraging partial in-memory data structures to alleviate both network and disk I/O. We call these modules **Glance** and train them for multi-spectral satellite imagery. Glance's novel approach reduces data transfers significantly by leveraging (1) model-based enhancement of image resolutions, and (2) dynamic mitigation of model error. Glance, a hybrid generative model, combines the concepts of progressive Generative Adversarial Networks(GAN) [12] along with image inpainting [101] to cope with unique characteristics of satellite imagery.

8.1 Overview of Framework

Glance comprises two key components -

1. **GlanceNet**

2. **Glance Error Mitigation Module (GEMM)**

8.1.1 GlanceNet

GlanceNet are a set of GAN-based deep neural network models that are trained to generate high-resolution data on-the-fly with high fidelity by using information available in the local STASH populated based on historical queries. Particularly, GlanceNet is designed for reconstruction of *multi-spectral satellite imagery*. GlanceNet inputs can be in a combination of both low-resolution and partial high-resolution tiles (if any). Our framework supports multiple levels of super-resolution (zoom) over multiple starting zoom-levels with high fidelity. For a requested im-

age at zoom-level $(n + u)$, i.e., 2^u times higher the resolution of an available cached image, all relevant tiles (resolution $(n + u)$) cached as a result of previous browsing, may also be leveraged.

GlanceNet comprises two sets of GAN-based components, each fine-tuned for a different scenario: the **Super-Resolution** and the **Image Refinement** models as depicted in **Fig. 8.1**. The Super-Resolution module is the core component of GlanceNet that handles the hierarchical geospatial super-resolution based on input images that may be available at different lower resolutions. The Image Refinement is an additional module that can enhance the output of the Super-Resolution module in case sufficient amount of partial high-dimensional data is also available.

The super resolution and image enhancement components work in tandem to generate imagery with high fidelity by enhancing the perceptual quality of super-resolution image. These components are leveraged selectively based on the number of image tiles that are available at the desired resolution in cache. GlanceNet is trained by independently modeling the Super-Resolution and Image Refinement components in parallel over our distributed storage system in a distributed data parallel mode to leverage data locality. Next, we chain these pre-trained modules and optimize the complete network. The trained network(s) is made available to the clients for download. They can be used in conjunction with a stand-alone STASH for use during interactive visualization.

Due to the large spatiotemporal extent of our data, we anticipate the trained GlanceNet models to have uneven performance over different regions of the overall data domain. To ensure sufficient accuracy for our reconstructed image, we propose **GEMM**, that underpins the dynamic decision-making process that informs the system's decisions regarding whether to retrieve the actual data from the back-end server.

8.2 Super-Resolution of Image Tiles

To ensure faster convergence and low memory footprint for our models, we harness incremental curriculum learning approach – leveraging knowledge gained from a lower-level SR network to train for higher SR to simplify the subsequent upscaling inference and speed-up the learning pro-

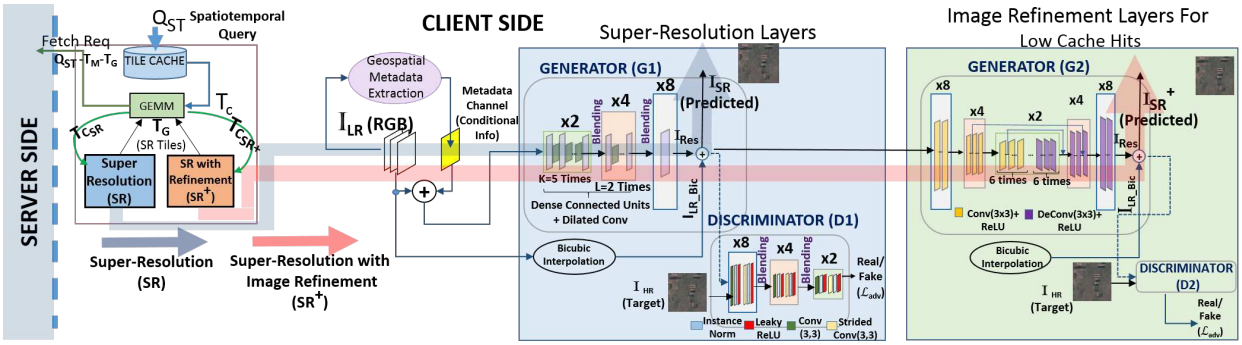


Figure (8.1) GlanceNet Architecture: Super-Resolution and Image Refinement layers along with the Glance Error Mitigation Module (GEMM)

cess. GlanceNet models also incorporate geospatial metadata into their input to provide conditional information to account for spatial variations in satellite imagery.

8.2.1 Model Overview:

The super-resolution (**SR**) component facilitates a hierarchical geospatial *multi-level* super-resolution operation by learning an upsampling function that transforms a low-resolution image tile ($\mathbf{I_{LR}}$) to generate a high-resolution synthetic rendition ($\mathbf{I_{SR}}$) that is contrasted against the ground truth ($\mathbf{I_{HR}}$). The SR component handles the primary scenario where a visualization client requests an image tile, which is not present in the in-memory cache, but a lower resolution parent of that tile is available in the tile pyramid. Additionally, the SR component accounts for scenarios where a fraction of the tiles comprising the target high-resolution image (i.e. Partial Super-Resolution Mode) is available in-memory, which is a common phenomenon due to the observed tendency of users to focus on a certain spatiotemporal region during a single visualization sequence [9] leading to persistent areas of interests through OLAP actions such as panning or drill-down.

In order to enable GlanceNet models to perform multi-level super-resolutions/ upscaling over a wide range of geospatial resolutions (zoom-levels), we have to take care of complications arising from the following challenges:

- Avoid building individual models for each specific super-resolution scenario that would overwhelm the combined memory footprint of the trained GlanceNet modules at the clients.
- Ensure relatively simpler model architecture for easier storage and evaluation.
- Handle diverse topographical characteristics in satellite imagery at different zoom levels.

To resolve these issue, we propose a conditional adversarial network that utilizes *progressive growing of GANs* with additional metadata extracted from $\mathbf{I_{LR}}$ as conditioning information. Using this approach, the sequential training of all the sub-models (x2,x4 and so on) captures the relations between all levels of cascading resolutions without going through disjoint, prolonged training phases.

8.2.2 Model Input

The input to our model is a composite image comprising of the low-resolution data for the missing regions and high-resolution data for the partial available regions. Given a low-resolution RGB image (I_{LR}), the input to the super-resolution network is constructed by super-imposing the available high-resolution image tiles, if any, with the bicubic interpolation of I_{LR} for the missing regions (this is the super-imposed image I_{LR+}), as shown in Fig. 8.2.

Every raw satellite image tile-set has additional metadata associated with their domain. We extract the following tile properties to **condition the inputs to both our generator (G_1) and discriminator (D_1)** –

1. zoom level(z) of I_{LR}, and
2. dominant land cover type (based on the National Land Cover Database (NLCD) codes [102]) in terms of I_{LR} pixels (e.g. urban, forest, barren etc.).

The extracted conditional information $g = (z, nlcd\_code)$ is embedded and then passed through a fully connected linear activation layer to resize the embedding to an extra 4^{th} metadata channel that gets appended to the 3-channel RGB image (I_{LR+}). This helps condition the GAN's training

Model Type	Upscale Factor	x2	x4	x8
SR^*	Using Supplemental Metadata Info	36.95	33.85	32.17
	Using only RGB Image	35.65	32.77	31.56
SR^+	Using Supplemental Metadata Info	40.52	37.46	35.22
	Using only RGB Image	39.55	36.08	34.99

Table (8.1) Comparing Model Performance and Image Quality (PSNR) With and Without Supplemental Metadata Information

using specific geospatial information regarding the region I_{LR+} represents. This metadata channel gets appended to the discriminator (D_1) inputs. Having the GAN network trained *conditioned on the geospatial metadata* associated with each input image tile helps the model learn location-specific topographical patterns that are uniquely associated with different spatial resolutions. We To help the model focus on the missing regions, along with the metadata channel, an additional channel to the input is included (5-channel input) which denotes a mask that identifies the low-resolution segments of the input image. During training, we randomly vary the fraction of missing high-resolution tiles between 0-1 to cover all possible scenarios.

8.2.3 Generator Network(G_1):

The generator output for our model with upscale factor u is:

$$\mathbf{I_{Res}} = \mathbf{G_1^u}(\mathbf{I_{LR+}}|\mathbf{g}) \tag{8.1}$$

Instead of learning the complex mapping between I_{LR} and I_{HR} for any given upscaling factor, the super-resolution problem is simplified to learning the residual, I_{Res} that needs to be applied to a bicubic interpolation (upscaled) of I_{LR} (eq.8.2). This helps the model utilize standard upsampling operations (eg. bicubic interpolation) and specialize in learning the difference between it and a high-resolution image [69].

$$\mathbf{I_{SR}} = \mathbf{I_{Res}} + \mathbf{bic}(\mathbf{I_{LR+}}) \tag{8.2}$$

The mapping between (I_{LR+}, g) to the true residual for I_{SR} becomes increasingly more complicated as the upscale factor increases. The progressive training of the models is a form of *curriculum learning* [103], where more complex upscaling models are built on top of trained models that handle relatively simpler task of a lower upscaling factor. Our model building starts from the smallest upscaling factor(x2) and adds more layers to the previous, trained layers and builds on them as we train for higher upscaling factors. This nested training approach, where we learn complex mappings in a nested manner through smaller, incremental steps helps reduce the overall training time for models of all possible upscale functions and has been known to improve the numerical stability of the model [12].

In order to maintain efficiency in a deep network and to facilitate the flow of weights between its layers, we adapt the concept of a densely connected network as introduced in DenseNet [104]. The principal element of our nested architecture is a Dense Block, organized in a sequence with a downsampling layer between each, to keep the output feature size between the layers in check. Each Dense Block consists of a sequence of layers which have direct connection from the output of the all its previous layers. This organization of the layers facilitates information flow over a deep network and facilitates in transmission of collective knowledge over the layers. We use a variation of the Dense Blocks, the Dense Connected Units [69] (see Fig. 8.1), that removes the BatchNorm component from the Dense Blocks' layers, when that component ceases to show improvements over a large number of training epochs.

We adopt an *asymmetric pyramid* (Fig. 8.1) structure for our generator layers. This means that the number of dense layers used in modeling is higher for lower level upscaling problems and are progressively less complex as we upscale further. Consequently, the model for a lower level super-resolution contains a higher number of cascading Dense Blocks, allowing for quicker computations and reduced memory-utilization (Fig. 8.1), since the lower-level SR models have to deal with a smaller input size. For our implementation with multispectral images, we have limited the maximum achievable super-resolution to x8 – beyond that, the generated satellite images cease to maintain their desired perceptual quality.

To achieve high accuracy in our super-resolution output, we target *improving the receptive field* of our model, which captures relationships between distant regions in the image. Although increased sub-sampling (downsampling) between layers (e.g., through max-pooling) is known to improve the receptive field, it also introduces blurriness in the image output, especially for higher upsampling factors. To help improve super-resolution operations for x4 and x8, we introduce **dilated convolutions** [105] between the Dense Blocks. This reduces the complexity of the model while improving the receptive field through the newly added layers (see Fig. 8.1).

Transfer of information from a lower upscaking model to the next upscaling model is performed through the process of gradual **blending** (Fig. 8.1). With blending, the output of the network is a weighted sum of the upsampled output from the previous (trained) model's layers (α) and the output from the current model ($1 - \alpha$), with α(influence of the previously-trained model in the pyramid) exponentially decaying and approaching 0 with each iteration.

8.2.4 Discriminator Network(D_1)

The discriminator layers (D_1) follow a similar, but significantly simpler, incremental structure as the generator layers, with more layers added with increasing upscaling factor. The organization of the layers on the discriminator mirrors that on the generator side. Instead of evaluating the entire output image, the discriminator(s) are designed to evaluate local topographical features for smaller geospatial areas. Similar to a PatchGAN [106] the discriminator produces an array as output, where each element evaluates a segment of the input image instead of the entire image; this helps model high-frequency details in the generated image. In our case, we have kept the receptive fields to 5x5, 11x11 and 23x23 for upscaling factors of x2,x4, and x8 respectively.

8.2.5 Objective Functions:

For our training, we have used two loss functions, optimized using an Adam optimizer [107]. The *reconstruction loss* in our case is the average pixel-wise difference between the generated and the real high-resolution image, **normalized** by the fraction of the masked area. Given a sample (of size N), for the i^{th} input for our model - (x_i, g_i, y_i), where the individual elements represent

the low-resolution input (of dimension CxWxH), its geospatial metadata and the corresponding high-resolution target, respectively, the reconstruction loss can be represented as:

$$\mathcal{L}_{rec} = \frac{1}{N \times C \times W \times H}[y_i - (bic(x_i) + G_1^u(x_i, g_i))]^2$$

The **adversarial loss** is computed as:

$$\mathcal{L}_{adv} = E[log(D_1^u(y_i, g_i))] + E[log(1 - D_1^u((bic(x_i) + G_1^u(x_i, g_i)))]$$

Both G_1^u and D_1^u are optimized using the adversarial loss of solving:

$$arg\ min_{G_1^u}\ max_{D_1^u} \mathcal{L}_{rec}$$

.

The **joint loss** is computed as

$$\mathcal{L} = \lambda_1(\mathcal{L}_{rec}) + \lambda_2(\mathcal{L}_{adv})$$

The generator is trained on this joint loss (with λ_1=1 and λ_2=1000), while the discriminator is trained to minimize the adversarial loss.

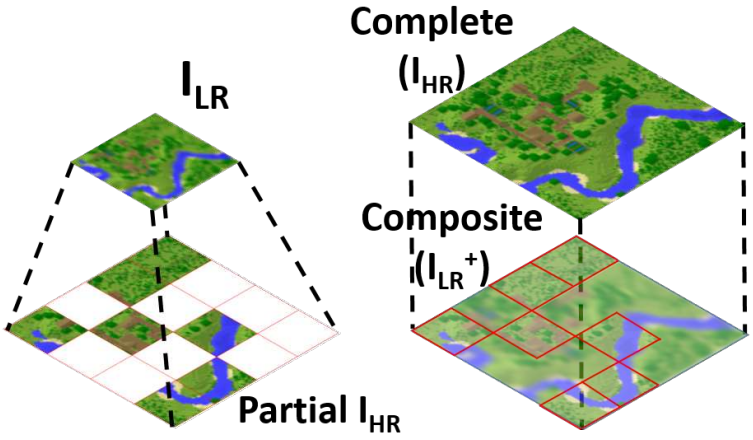


Figure (8.2) Composite Input for Partial Super-Resolution

8.3 Image Refinement

The Image Refinement network is an additional module that can help further refine the output of the Super-Resolution layers by utilizing high-resolution partial data tiles.

8.3.1 Model Overview

This model acts as a refinement layer for cases where a significant fraction of the high-resolution image tiles is available in cache - this fraction is configurable and we set it at at-least 50% for our study. This scenario is analogous to a *simplified inpainting problem*, where, given the composite image with blurry regions, I_{LR+} (Fig. 8.2), we have to deblur these regions for high-resolution information.

8.3.2 Model Input

The input to the network is the composite image (I_{LR+}) which is constructed by superimposing the available high-resolution image tiles (50-100% randomly selected) over either the bicubic interpolation of I_{LR} for the missing regions during the pre-training and with I_{SR} (output of the Super Resolution Network) during the chained model training phase.

8.3.3 Generator (G_2)

The generator layers (Fig. 8.1) comprise deep fully-convolutional encoder-decoder network with an equal number of convolution and deconvolution layers with symmetric skip connection between corresponding convolution and deconvolution layers every third layer, as introduced in [108]. The convolution layers enable abstraction through feature extraction from the image, while the deconvolution layers perform image regeneration, thus eliminating/ reducing noise. The skip connections facilitate effective information flow over all the layers, particularly in the higher upscaling layers.

We further enhance this encoder-decoder network for our *curriculum learning* approach introduced in Sec. 8.2 by introducing additional convolutional and deconvolutional layers incrementally as we train higher upscale factors along with blending of the new model layers to ensure smooth

Model Type	Upscale Factor	x2	x4	x8
SR	Using Supplemental Metadata Info	36.95	33.85	32.17
	Using only RGB Image	35.65	32.77	31.56
SR^+ **(SR + Image Refinement)**	Using Supplemental Metadata Info	40.52	37.46	35.22
	Using only RGB Image	39.55	36.08	34.99

Table (8.2) Comparing Model Performance and Image Quality (PSNR) With and Without Supplemental Metadata Information

transition between successive models and faster convergence. All other components of the Image Refinement Network have the same configuration as the Super Resolution Network.

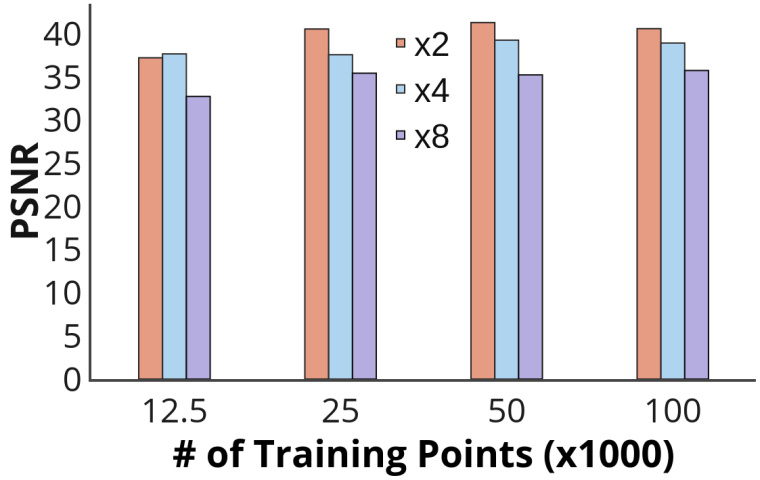


Figure (8.3) Super Resolution with Image Refinement ($\mathbf{SR}^+$)

8.3.4 Evaluating Image Quality

Table. 8.2 demonstrates that including of geospatial metadata as conditional information to our models has a positive effect on the generated output of the super-resolution models across various upscale factors. Additionally, we also see that including the Image Refinement layers also increases the PSNR of the generated outputs.

Fig. 8.3 demonstrates the fidelity of the generated images using the Super Resolution along with the Image Refinement module for different levels of super-resolution. We also see that an increase in number of training data helps in the quality of the generated outputs.

8.3.5 Upsampling Image with Low Cache Hits

For a cold-start scenario, where a client cache has no previously fetched tiles, the generalized scenario of interpolating the input to I_{LR+} along with a channel denoting the partial high-resolution information mask becomes redundant.

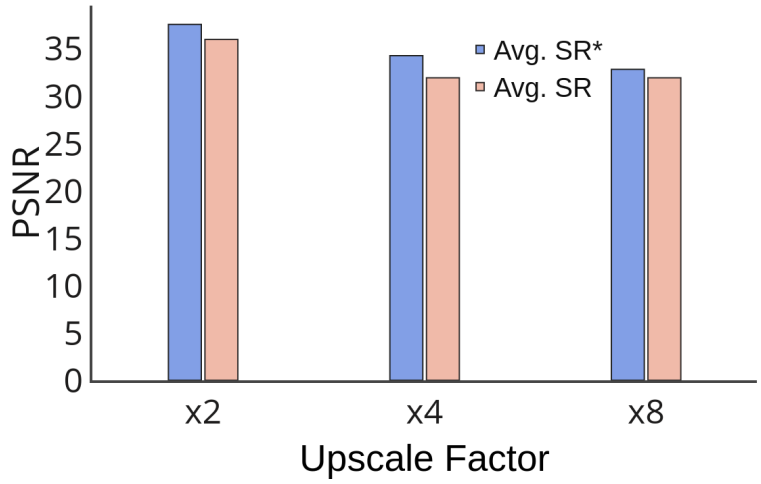


Figure (8.4) Cold Start with SR^* vs SR

We have designed an optional Basic Super-Resolution Model dedicated to such cold-start scenarios that use a 4-channel input of (I_{LR}, g) instead (let us call this the **basicSR-GlanceNet** mode). This lower input-size of the model allows us to increase the complexity of its layers to improve the modeling of the upscaling function in light of no partial high-resolution data. Although the structure of the layers is still the same, the values of K and L for G_1 in Fig. 8.1 are increased to 5 and 2, respectively. We expect this specialized model to generate images with higher perceptual quality for such cold-start scenarios. Thus, for systems that demand higher-quality images at the front-end, clients can choose to download optional component of GlanceNet.

Fig. 8.4 shows that using the cold-start model(SR^*) instead of the general-purpose GlanceNet (SR^+) has a positive effect in the image quality in cold-start scenarios where no partial HR image tiles are available in-memory. However, due to the more complicated architecture of the cold-start model, it would have an effect on the image-reconstruction latency, as we show below.

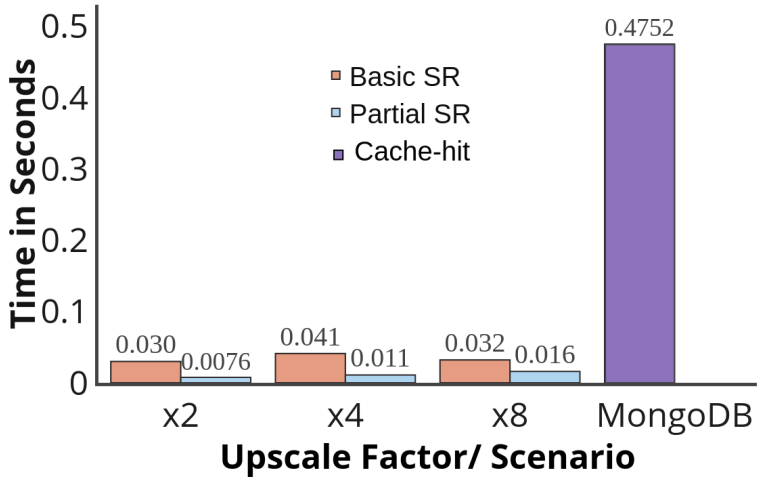


Figure (8.5) Client-side Average Image Reconstruction Time vs Cache Fetch Time

8.3.6 Improvement in Latency

Fig. 8.5 demonstrates the average client-side latency to generate a high resolution image in-house and compares it to the time taken to fetch from a cache-enabled distributed server (MongoDB). We see that *in-situ* image reconstruction is significantly faster than a fetch from server, which is expected due to the additional network communication and cache-lookup involved. We have also compared the average time taken by the GlanceNet models to reconstruct (with GPU acceleration) a set of 100 images with a batch-size of 16 for varying upscale factors. We can see latency in the order of milliseconds for all upscale factors for both GlanceNet super-resolution modes basic-SR and SR^+ (partial SR). SR^+'s latency are relatively lower than SR^* due to simpler model layers.

8.4 GEMM: Estimating Regional Model Accuracy

The large spatiotemporal extent of our datasets introduces high variability in the input images that are exacerbated by variable cloud cover, image noise and other natural phenomena that might adversely impact the quality of images from a spatiotemporal region. Additionally, single image super-resolution GANs do not guarantee accurate results, especially for satellite imagery with higher upscaling factors. The loss of high-frequency details such as architectural details in dense urban areas or regional cloud coverage leads to blurry/noisy outputs. Although maintaining individual regional models might help improve the quality of the results, it is infeasible to maintain such a high number of models at the client.

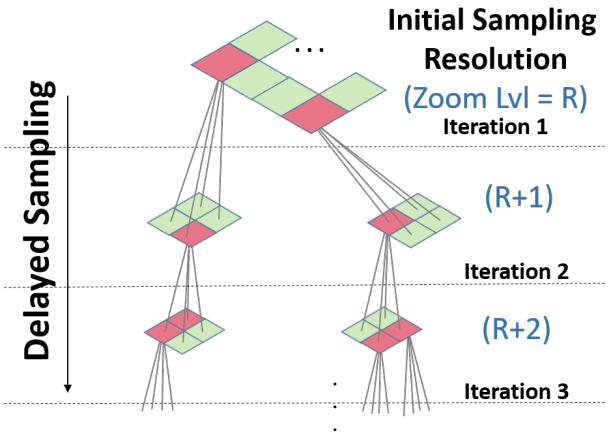


Figure (8.6) Glance Error Mitigation Module

To mitigate this variability in model performance, Glance's GEMM module tracks regional accuracy variations based on the geospatial characteristics. GEMM can be fetched and stored in the clients' memory. This regional model accuracy is a key element to be considered when an application determines whether to download the original image or to use the model inferred output. Additionally, GEMM helps monitor the overall model accuracy that can trigger retraining of the GlanceNet modules on crossing a threshold.

GEMM is created server-side, after the training of the global GlanceNet in an evaluation phase where the samples are generated initially (first iteration) in a uniformly spatiotemporally parti-

tioned manner. Glance estimates the regional model accuracy as the average of errors across sampled images for that region. These accuracy estimates are stored in a server-side hierarchical in-memory spatiotemporal tree structure (Fig. 8.6). Each node represents a geospatial region and stores the corresponding estimate. The geospatial hierarchy of this data structure is aligned with the structure of underlying distributed cache so that the data and regional accuracy can be mapped directly during query evaluation.

Given the large spatiotemporal coverage and the number of resolution levels, maintaining a fully populated data structure is not effective. To address this, Glance **dynamically refines the geospatial domain for sampling** to focus on regions with higher variation in errors iteratively. The geospatial coverage of the sampled partition starts from a large area (with a coarse resolution) and is incrementally refined as the error variation metrics of the given partition exceeds thresholds. Each incremental refinement adds child nodes to the current node in the data structure depicted in Fig. 8.6. The final data structure may not be a fully-balanced tree, but this strategy effectively reduces the memory-footprint for the large dataset.

Our system continues to refine the geospatial coverage until the average accuracy is sufficiently close to the individual accuracies from the samples without higher variations. Finally, the error variance, e_v is calculated as the disparity of errors using the Normalized Root Mean Squared Deviation.

$$e_v = \frac{1}{Q_1 - Q_3}\sqrt{\frac{\sum_{i=1}^{n}(e_i -)^2}{n}} \tag{8.3}$$

,where $Q_1 - Q_3$ is the inter-quartile range and the rest of eq.8.3 corresponds to the sample's root-mean squared error (RMSE)

8.4.1 Improvement in Image Quality vs Query Latency

As explained in previous evaluations, although the identification of low-performing geospatial domains and performing physical fetches for them is beneficial to the overall output quality, it also increases the query latency. GEMM's error threshold has an effect on the overall fraction of image

	x2				x4				x8			
	Thresh	% Fetch	PSNR	Look-up	Thresh	% Fetch	PSNR	Look-up	Thresh	% Fetch	PSNR	Look-up
SR	0.02	1.2	38.18	0.13	0.02	0	33.55	0.06	0.02	0	32.33	0.02
	0.018	12.5	40.87		0.015	0	33.48		0.015	0	32.78	
	0.015	36.25	41.82		0.01	35.93	36.75		0.01	14.5	32.98	
	0.01	76.75	47.95		0.008	52.08	37.55		0.008	37.5	34.32	
SR$^+$	0.04	8.41	45.71	0.11	0.03	10.5	38.67	0.09	0.025	2.08	36.52	0.02
	0.035	12.33	46.63		0.01	22.91	40.04		0.005	12.5	38.87	
	0.03	31.25	48.16		0.006	42.08	47.38		0.003	41.25	46.34	
	0.025	57.91	50.43		0.004	58.33	51.41		0.002	81.25	53.84	

Table (8.3) GEMM Performance Evaluation

tiles fetched from the server and thus an adverse effect on its latency, as shown in Table 8.3. This demonstrates the need for setting a balanced threshold for the system.

Chapter 9

Model-Driven Extraction of Insights from Embeddings (ARGUS)

The ARGUS framework is designed to address rapid analytical keyword query evaluation over voluminous multi-spectral satellite imagery. Each of these data-objects have high spatial resolutions and multiple data bands, making them large in size. Caching them in their original form would significantly strain cache capacity and negatively impact the hit-rate and the interactivity of spatiotemporal query analytics. Our goal is to ensure high fidelity for model-driven analytical information compared to geoprocessing over actual data objects. Scalable management of voluminous data collections [109, 110] underpins effective training of deep networks [18]; data accesses are also predicated on effective queries [111] and federation [112]. Several systems also rely on outlier detection [113] to preferentially identify training data of interest.

Useful analytical information can be extracted from these multi-spectral images using geoprocessing algorithms such as the computation of slopes, and curvatures from raster images using spatial tools provided in frameworks like ArcGIS [114]. However, these algorithms are often not optimized for parallel or GPU-driven execution, making them significantly slow, especially when the number of candidate tiles required to be processed is large. This is particularly true in cases of sparse events like wildfires.

For instance, the following SQL query provides a sample of the type of spatiotemporal queries that the ARGUS framework aims to evaluate for multiple simultaneous users. In particular, we show a wildfire segmentation query for identifying and demarcating potential wildfire regions (*has_fire*) from multi-spectral satellite imagery dataset (VIIRS) over a given spatial and temporal range specified through the *Query_Polygon* and *Query_Time_Range*, respectively.

```
select has_fire(band_1, band_2,..., band_n)
from VIIRS_Data
```

```
where coordinates in Query_Polygon
and time_stamp in Query_Time_Range
```

The evaluation of the above query over a distributed storage system would involve identifying image tiles with intersecting spatiotemporal bounds, evaluating their wildfire-affected regions, if any, and returning the compiled results back to the users. ARGUS attempts to speed up the above process by implementing the following - (1) the creation of embeddings out of image tiles for easier storage in a distributed cache for rapid identification, (2) training deep-learning analytical models that use embeddings as input, circumventing the need for on-disk data access as well as geoprocessing algorithms, (3) using a combination of classification models for identification of potential tiles with wildfires and running segmentation model on those only, and (4) using efficient in-memory caching and indexing schemes to avoid both disk access and re-evaluation of embeddings. Since events like wildfires are sparse, running a simpler classification model to weed-out unnecessary tiles can significantly improve interactivity.

9.1 System Components

ARGUS is designed to work in conjunction with any distributed hash table (DHT)-based spatiotemporal storage system [83]. The overall ARGUS framework can be partitioned down into two main components -

1) ARGUSNET: A collection of deep learning models trained to perform encoding and keyword-based evaluation from the unlabeled satellite data collections, and

2) Hierarchical Embedding Store: A graph-based in-memory caching framework built to house latent representations generated by the ARGUSNET module.

Fig.9.1 shows the various components of our framework. The ARGUSNET models utilize data from the underlying DHT storage for their training through distributed modeling. Once trained, the ingestion module utilizes the encoder portion of the network to intercept data ingestion requests and house them in the hierarchical embedding store. The classification and segmentation models are used during query evaluations over the cached latent representations in the embedding store.

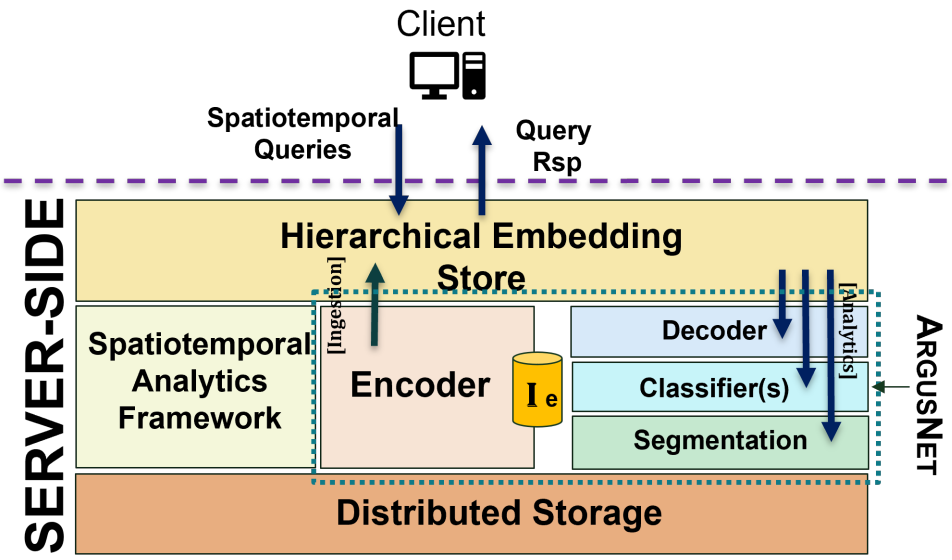


Figure (9.1) ARGUS System Overview: Hierarchical Embedding Store is our distributed in-memory caching system. Encoder, Decoder, Classifiers and Segmentation constitute the various components of ARGUSNET

9.2 ARGUSNET

9.2.1 Model Overview

Ideally, the latent representations of our image tiles must be versatile enough to support multiple keywords (e.g., occurrence of wildfire and the level of severity) without maintaining multiple embeddings for each problem. To accomplish this, we train the models in a conjoined manner through multi-task learning and generate a single embedding for each tile that is used for multiple analytical models later on. Related tasks trained through multi-task learning have been shown to have better accuracy and convergence speed and as evidenced by our benchmarks. Our models demonstrate improved accuracy as well. Fig.9.2 depicts the overall model architecture. We can see that it consists of the following main components – an **autoencoder** network, **classification** networks, and a **segmentation network**. Additional models for the extraction of related analytical information from the embeddings can be added to our network as needed. Apart from the autoencoder network, all other networks (heads) use embeddings as their input.

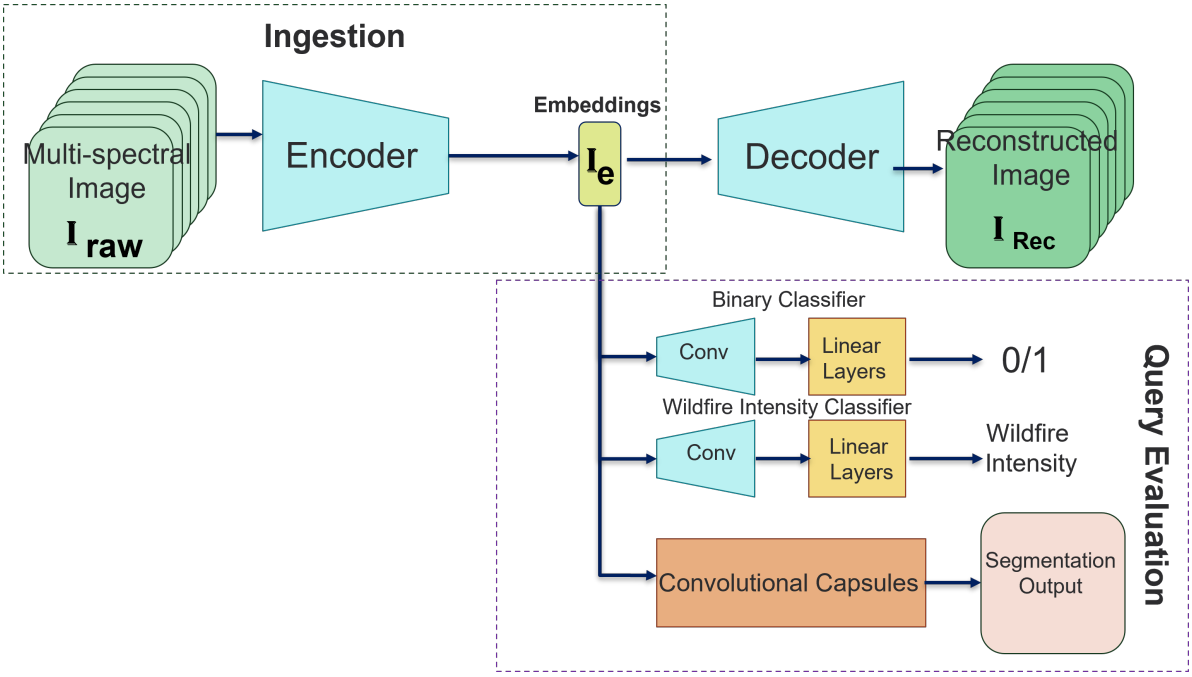


Figure (9.2) ARGUSNET Architecture: Encoder forms the backbone of the network used during data ingestion to generated embeddings. Decoder, Classifiers, and Segmentation heads are used during query evaluations.

9.2.2 Model Input

Let us denote our multi-spectral image input as $\mathbf{I_{Raw}}$ - this is the input to our ARGUSNET network, which gets converted to its corresponding latent representation, denoted by $\mathbf{I_e}$, which is significantly smaller in size. In our case, $\mathbf{I_{Raw}}$ is complied by integrating various distributed spatiotemporal datasets and extracting relevant bands from them that are relevant to wildfire prediction. The target band, along with the classification labels is extracted by combining the fire-band available in the VIIRS dataset [115] along with historical wildfire perimeter (and duration) information to create a single-channel target mask for each image tile ($\mathbf{I_{Target}}$).

9.2.3 Selection of Training Data:

Our input is created through a combination of multiple remote-sensing data sources and includes bands relating to emissivity, soil moisture, vegetation index, and land cover type, all of

which are known to be contributing factors that influence wildfire. In total, our input, $\mathbf{I_{Raw}}$, consists of 15 bands/channels.

Class imbalance is common in case of wildfires since the majority of the image tiles will not contain fire-pixels. In order to circumvent models prioritizing the majority class, we oversample the wildfire-containing tiles. We use the California Fire Perimeter Database [116] for historical information on wildfire perimeters and dates to identify tiles that have fire pixels in them. Additionally, to reduce the uncertainty in the training data, we ignore tiles that contain wildfires with an overall perimeter area of more than 10 km^2. Finally, we use a 1:1 distribution of fire and non-fire tiles in our training.

9.2.4 Network Architecture:

Our goal is to perform semantic segmentation through convolutional networks for the detection of wildfires from multi-spectral imagery. ARGUSNET consists of two main stages: a set of convolutional layers for feature extraction and a set of heads for performing reconstruction, classification, and segmentation. We explain each section of the overall deep learning model below.

Encoder: The encoder constitues the backbone of the ARGUSNET architecture. In the first stage, this encoder portion of our autoencoder, comprising a set of convolutional layers, generates a dense representation of a multi-spectral image tile. We expect these convolutional layers to take a multi-spectral image vector as input and encapsulate complex and abstract features from the input image for analytics. This extracted feature map serves as an input to the three heads of the ARGUSNET network.

The encoder network comprises a series of convolutional layers followed by a downsampling through max-pooling that incrementally reduces the spatial dimension while increasing the number of channels leading to bottleneck. We introduce batch normalization between the two layers to stabilize the training process to avoid bias during training by normalizing the input to each layer and accelerating the convergence speed of the training. The output of the encoder network produces our embeddings (I_e), a compressed representation of the abstract features of the input image (as

shown in Fig.9.2).

The main goal is to be able to utilize the generated I_e for the extraction of multiple analytical observations. In order to make it versatile enough, we have to ensure that the training process takes into consideration the loss of each of these model-driven analytical tasks during the construction of the embedding and not just the reconstruction loss. **Multi-task learning** (MTL) is an effective approach to training and optimizing a combined model to perform multiple tasks simultaneously. This conjoined training methodology, where a cumulative loss from all the related tasks affects the weights of the network, enables the model to leverage shared information between tasks. This has been shown to produce better representation learning, regularization, transfer learning, and improved data efficiency. Multi-task learning can improve the accuracy of all models in several ways, including the ability to learn more general representations of the data, prevent overfitting, and facilitate the reuse of learned features for related tasks. Overall, training our models through the combined architecture, as shown in Fig. 9.2, can significantly improve the accuracy and generalization of machine learning models.

Decoder: The compressed latent representation is passed on to the decoder network, which uses upsampling of feature maps through a series of transposed convolutional layers (deconvolution) and increases the spatial dimensions of the data to eventually reconstruct the input ($\mathbf{I_{Rec}}$). Maintaining a decoder head trained for image reconstruction serves two purposes. First, it allows us to use the embedding to recreate the bands of the original image in case of queries over the actual bands. Secondly, it allows us to introduce new heads into the network while ensuring faster retraining convergence speed.

Classifier: The classifier heads are responsible for generating a probability distribution over either a binary flag that predicts whether an image tile has wildfire, or over the possible wildfire intensity classes. The classifier head takes the latent representation, I_e, as input and flattens it into a 1D vector. This vector then passes through a set of fully connected layers to produce a vector of scores, one for each object class. A softmax activation is applied to these scores to generate a probability

distribution over the classes.

Semantic Segmentation: We implement a SegCaps [117] architecture as the deep semantic segmentation head for our network. We leverage capsules, which are a variation of a neuron or *instantiations* that represent a specific aspect of the object and can encapsulate various properties of an object, including, its spatial orientation, and scale. This feature of capsule network helps us adapt to wildfires of different geospatial scales and additionally leverages the capsule's ability to understand relative positions and orientations of objects in an image to train on wildfires which also have regional characteristics. Similar to [117], our network also contains a set of 8 primary capsules, followed by digit capsules as the output layer that generates the segmentation output. Another important benefit of using capsule networks for wildfire segmentation is their ability to reduce the number of labeled datapoints for training, which is useful for wildfire events, that tend to be pretty sparse.

9.2.5 Loss Function:

Our multi-task loss function is a combination of losses from the heads of the ARGUSNET network. We explain each of these components below.

Reconstruction Loss: This is the loss component from the decoder head. Since we mainly want I_e to be useful in extraction of model-driven analytical information, we prioritize minimizing the reconstruction loss (Mean Squared Error) of bands that are more closely correlated to wildfires, such as NDVI, land-cover type, and soil-moisture. We update reconstruction loss and give more weight(W_1) to the bands with more correlation(B_1) to wildfire than the remaining bands(B_2), i.e., $W_1 > W_2$:

$$\mathcal{L}_{rec} = W_1 * \mathcal{L}_{rec}(I_{Rec}[B_1]) + W_2 * \mathcal{L}_{rec}(I_{Rec}[B_{Rem}])$$

Classification Loss: Our classification loss ($\mathcal{L}_{\mathbf{class}}$) is computed as Cross Entropy loss. For predicting the presence of a wildfire in a tile, we train using Binary Cross Entropy Loss (BCE) and for the multi-label prediction of fire-severity, we use BCE with Logits loss (combination of a Sigmoid layer and BCE Loss).

Segmentation Loss: Wildfire-affected regions can comprise varying portions of the pixels in an image. In case of smaller fire perimeters, we ensure that we avoid a biased segmentation model that prioritizes classifying the background pixels. Dice loss has been shown to be suitable for such class-imbalance problems [118]. So we make our overall segmentation loss ($\mathcal{L}_{\mathbf{seg}}$) a combination of the BCE loss and the Dice Loss.

The overall loss of the network is computed as follows:

$$\mathcal{L}_{\mathbf{rec}} = \mathbf{W_R} * \mathcal{L}_{\mathbf{rec}} + \mathbf{W_C} * \mathcal{L}_{\mathbf{class}} + \mathbf{W_S} * \mathcal{L}_{\mathbf{seg}}$$

The weights of each loss, i.e. W_R, W_C, W_S are hyperparameters that we optimize during the training process.

9.3 Distributed Training

Our ARGUSNET module is trained over a distributed spatiotemporal filesystem. The server-side cluster tracks the freshness of the trained models with respect to new, incoming data and triggers a fresh round of training iterations to further fine-tune old models.

The training of the models is agnostic of the underlying distributed file system. Our training leverages the spatiotemporal partitioning scheme of the storage system by collocating the training modules with the partitioned data to avoid data movements during training. The distributed training utilizes a parameter server to aggregate the weights asynchronously at certain intervals. We have used Pytorch Lightning [119] in the Distributed Data Parallel (DDP) mode for our distributed learning. Once trained, we use the encoder part of the network for our ingestion processes, while the classification and segmentation branches are used during query evaluations.

Fig.9.3 shows the rate of convergence of our MTL setup with encoder output connected to the decoder, segmentation, and classification heads. We profile the segmentation loss, which is a combination of Binary Cross Entropy and Dice Loss for our predicted wildfire-affected area (mask) for both our training and validation data.

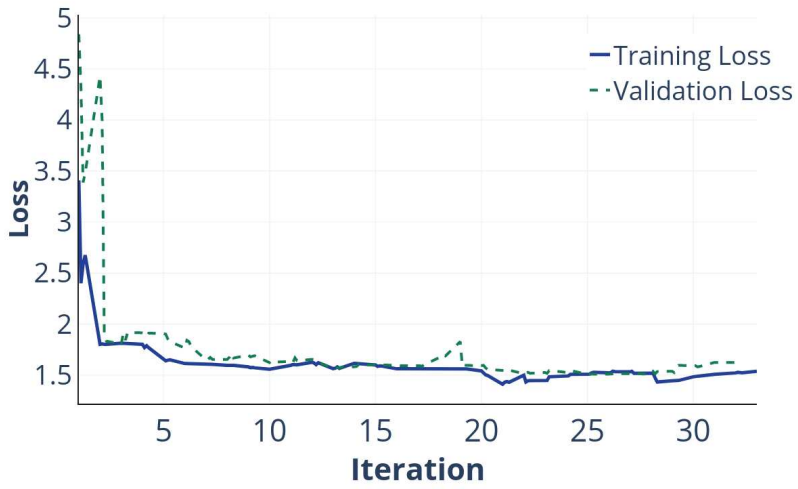


Figure (9.3) Convergence Speed of Model: Variation of training and validation error for ARGUSNET over epochs.

Table (9.1) Comparison of ARGUSNET Evaluation Performance against Standalone Segmentation Model

	Convergence Time(minutes)	**Epochs**
ARGUSNET	6.31	31
SegCaps	155.8	164

We compare the performance of our multi-task learning setup with a Classifier, Decoder, and Segmentation head against that of the dedicated SegCaps modeled after [117]. We can see, from Table 9.1 that the model stabilizes at around 31 iterations. The convergence of the SegCaps model is significantly slower, around 155 minutes and 164 epochs, which can be attributed to both the larger input size and the number of model parameters involved – the **total number of optimizable parameters** for our MTL setup was $\mathbf{47.35}\%$ that of a standalone SegCaps model for image segmentation.

9.4 Hierarchical Embedding Store

The **Hierarchical Embedding Store** is a lightweight indexing scheme to better organize the in-memory latent representation of the on-disk images tile on each node in the distributed cluster. The embedding store is a decentralized in-memory metadata graph that holds the embedding ob-

ject, $\mathbf{I_e}$, in its leaf nodes, along with other information that gets populated with subsequent query evaluations, such as, predicted fire-masks and flags marking the presence of wildfire pixels in the corresponding tile. The graph is organized based on the spatiotemporal metadata of the underlying tile, as shown in Fig.9.5 to facilitate fast query evaluation and identification of relevant in-memory embeddings for each node.

In addition to the Hierarchical Embedding Store, ARGUS maintains the trained modules, mentioned in the previous section, in-memory on each node for fast generation of embeddings (during ingestion) and for model-based query evaluation (during runtime).

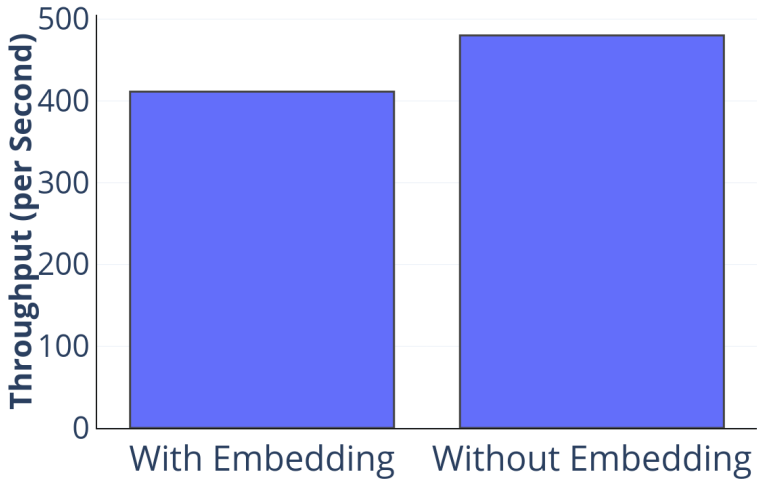


Figure (9.4) Ingestion Throughput With/Without Embedding: Comparison of throughput of indexing the in-memory metadata graph with and without generation of embeddings through encoder during ingestion.

Fig. 9.4 shows a further breakdown of the indexing process. Here we index and load a total of 5000 image tiles into our distributed in-memory graph and evaluate the overall throughput on each node. Here, we show the overhead of the encoding process that creates embeddings for each incoming image tile and stores a reference of the embedding object on the in-memory graph. Our encoder is simple enough and combined with the batched computation of embeddings, we can see that the indexing throughput is only reduced by 16.7%.

9.5 Data Model and Query Evaluation

Here, we explain the various stages of data ingestion into our embedding store and the subsequent process of querying analytical information out of it.

9.5.1 Data Preprocessing and Partitioning

Our input data is curated to incorporate attributes that are known to influence the likelihood and impact of wildfires in a region. In doing so, our approach leverages science-guided machine learning in our modeling for wildfire detection and segmentation. Science-guided machine learning [120] combines domain knowledge and scientific principles to enhance interpretability, reliability, and generalization of the trained models. In order to incorporate prior scientific knowledge into our model building, we integrate multiple data sources containing remote-sensing satellite data that provide global, near real-time information that includes active fires, thermal anomalies, and the Normalized Difference Vegetation Index (NDVI), which is computed from the red and near-infrared (NIR) bands of the VIIRS sensor [115]. Additionally, we incorporate relevant attributes such as land-cover type, soil moisture, and water retention, which are scientifically correlated with wildfires. By integrating these attributes, our model improves the accuracy of wildfire detection. The target segmentation mask is created by intersecting the active fire band from the remote-sensing data with historical wildfire perimeter information. For this study, we use the wildfires in California during 2019 [116] as our use-case.

The raw data once downloaded and merged needs to be partitioned for efficient storage, distribution, and querying over a cluster. We split each multispectral image swath in terms of its geospatial hash (9-character quadtile key) for efficient indexing. These image tiles are then partitioned over the cluster based on their temporal metadata and quadtile key.

Quadtiles [121] is a hierarchical grid system that can recursively partition the overall geospatial coordinate space, incrementally, into a set of squares, each divided into four sub-squares of equal size. Each sub-square is assigned a unique code, which is appended to the unique code of the cumulative square, forming a unique hash string that represents the geospatial region contained

Table (9.2) Breakdown of Data Staging: Comparison between time taken to download and pre-process the data against time to index and load it into ARGUS

	Fetch	**Processing**	**Staging**	**Indexing**
Time (Seconds)	132.07	539.53	303	**2.04**
Percentage	13.52%	55.24%	31.02%	**0.2%**

within it. This quadtile key can be easily manipulated to identify both neighboring geospatial quadtiles/regions, along with encapsulated sub-regions.

In this work, we use the entire coordinate space of California as the overall geospatial region, represented by a single square with a key of "0", and recursively partition them into incrementally smaller quadtile boxes, each divided into four sub-squares, and appending each with either "0", "1", "2", or "3". The generated tiles are distributed among the cluster nodes based on their quadtile key and within each node, are organized based on their date and time of recording. This distribution scheme helps the zero-hop DHT efficiently identify relevant tiles both during training and query evaluation.

9.5.2 Embedding Store Population

The lightweight indexing scheme of the Hierarchical Embedding Store enables fast population of entries. During data ingestion, each incoming image tile, before being stored on disk, gets converted into its low-dimensional latent representation, $\mathbf{I_e}$, through the encoder module (E), and stored in-memory. The spatiotemporal information of the tile is used to add it to Hierarchical Embedding Store, with the reference to the in-memory $\mathbf{I_e}$ object being added to the leaf-node. This ensures co-location between the on-disk data objects and their latent counterparts and the embeddings follow the partitioning scheme of the underlying distributed system.

The creation of latent embeddings using the encoder, along with its population into the hierarchical embedding store constitutes the computation overhead during data ingestion. ARGUS ensures that this computational overhead is insignificant compared to the actual ingestion process by (1) ensuring that the tree-based structure of the embedding store facilitates fast indexing, (2) the

encoder-decoder network is kept relatively shallow, and (3) the multiple incoming tiles are ingested as batched inputs to the encoder network for faster computation.

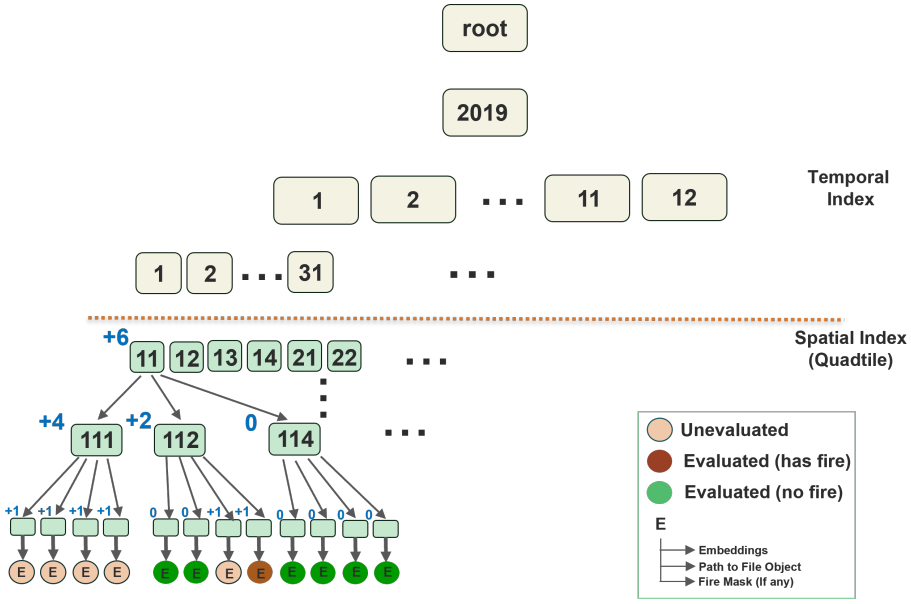


Figure (9.5) Hierarchical Embedding Store

9.5.3 Containerized Data Ingestion

Since our query analytics framework relies on the effectiveness of in-memory embedding cache, along with the trained deep-learning models, we need to ensure that a sufficient amount of memory and computational resources is available at all times. We ensure that our data ingestion process, which also requires GPU for the encoder network, does not adversely affect the query analytics.

In order to ensure a scalable ingestion throughput, while maintaining an upper bound on resource utilization, we parallelize and containerize our data ingestion processes. Ingestion requests for each node in the distributed system are inserted into a job queue from which they are handled by one of our available ingestion processes. We use Kubernetes [122] replica sets to ensure parallelization. We set a limit on resource utilization by configuring maximum resource limits on the

overall utilization of our data ingestion containers. The threshold is set at 10% of the overall CPU, memory, and GPU cores.

9.5.4 Query Evaluation

The Hierarchical Embedding Store enables fast identification of relevant embeddings for each spatiotemporal query on each node of the distributed cluster. In a cold start scenario, the graph is evaluated against the spatiotemporal bounds of the query in a top-down fashion to identify the latent embeddings that satisfy the specified predicate. These embeddings are meant to be probed for potential wildfire regions using our trained models. However, exhaustively evaluating all candidate tile embeddings against our segmentation mode, which has the most complicated architecture, for a sparse event such as wildfire might lead to prolonged response times. Rather, ARGUSNET maintains a simpler binary classification model for the identification of wildfires, which is first run to identify potential embeddings (containing wildfire) that get evaluated against the segmentation model. As we show in our benchmarks, this strategy leads to significantly reduced response times with comparable accuracy. Additionally, similar to the case of the encoder, the evaluation of embeddings for both classification and segmentation models is done in batches.

9.5.5 Avoiding Redundant Evaluations

Geospatial access patterns have been shown to follow spatial and temporal locality [9], i.e., at a given instant, users' queries over the entire dataset are focused on a small spatiotemporal neighborhood. While effective caching can leverage these patterns and greatly improve the hit-rates of our in-memory structures, this does not avoid redundant evaluation of our embeddings against the classification and the segmentation model, which require GPU resources.

The structure of our Hierarchical Embedding Store makes it conducive to support simultaneous query evaluations and collaborative analytics. Since it is structurally a feature graph, it can easily be traversed in a top-down manner while evaluating a spatiotemporal query. Fig.9.5 demonstrates our hierarchical strategy of tagging potential wildfire nodes in the Hierarchical Embedding Store.

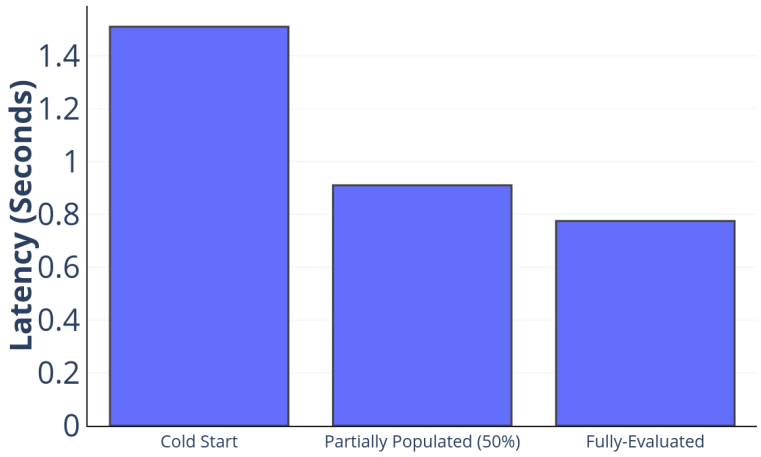


Figure (9.6) Improvement in Query Latency with cached evaluations from historical queries.

Upon evaluation of a tile against a spatiotemporal query, if a non-zero fire-mask is returned by our segmentation model, we store a compressed representation of the 2d array along with the embedding object in the leaf node. When no wildfire is detected, it will have a blank object and all unevaluated tiles have a null object attached to their leaf. This helps our framework avoid redundant computations of the same image tile against our trained models when there are subsequent overlapping requests. Additionally, it is to be noted that the memory overhead of having this fire-mask info is quite low - given the sparsity of the event, very few nodes will actually require the fire-mask object to be stored.

9.5.6 Optimized Graph Evaluation

Due to the large number of tiles (leaf nodes) that might be involved in queries over large spatiotemporal extents, we attempt to identify parent nodes that do not have any tiles of interest in their sub-tree. We keep track of these nodes through successive query evaluations over our emdedding store by maintaining an additional attribute (**node importance**) on each node of the Hierarchical Embedding Graph. This attribute signifies the combined number of unevaluated and wildfire tiles under each parent node in the graph - a non-zero importance value means that during

evaluation, a parent node may contain a tile of interest in one of the leaves in its subtree. Over time, with a meaningful number of queries being evaluated over our system, a majority of these tags should amount to 0 (since wildfire is a sparse spatiotemporal event), which would help us avoid unnecessary traversals down the graph for irrelevant spatiotemporal extents.

Fig.9.5 demonstrates the update strategy of node importance during the evaluation of queries over ARGUS. When the segmentation output on a tile embedding is found to have no fire pixels, the node importance of its immediate parent is decremented. If the count of the parent is 0, we decrement the count of its immediate parent, and so on, up the graph. In general cases, this upward traversal of a tree would require bidirectional links or extra computation. Since our Hierarchical Embedding Store is a metadata graph, actual upward traversal up the tree can be avoided, since, given the spatiotemporal metadata of a node, we can easily deduce the exact parent node. In successive query evaluations, any node with importance of 0 will not need to be traversed further since it signifies non-fire tiles underneath.

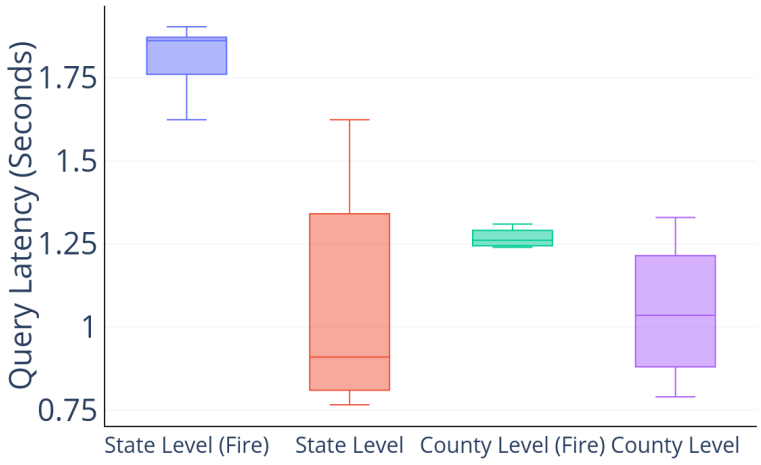


Figure (9.7) Query Latency vs Query Size: Evaluation of increase in latency with the scale of the query's spatiotemporal extent.

We demonstrate the scalability of our model-driven query evaluation in Fig.9.7. We profiled the average query latency for state and county-level spatiotemporal queries for 2 scenarios - one over spatiotemporal regions where we know had wildfires and second over random spans and regions. We execute 1000 different queries over our cluster and evaluated the average response time at a client node. As expected, state-level queries take longer to evaluate than county-level queries, but the average query time is reasonable. Additionally, we can see that in fire-prone scenarios, the query latency is higher than in average case scenarios, since the number of tiles that are actually subjected to the segmentation model is significantly low, due to the classifier model filtering them out. The box-plots in Fig.9.7 show that a majority of the queries have significantly lower query latency in the average case since the majority of the spatiotemporal queries have no wildfire in them and most of the server-side overhead is simply from evaluation and classification over the in-memory metadata graph.

9.5.7 Pruning: Node Replacement Strategy

Due to the limited size of the cache, we need an efficient strategy to maintain frequently accessed and relevant information in the cache in case of overflow. In order to facilitate interactivity in evaluations over the metadata graph, we need to maintain the most relevant regions in memory, and to efficiently detect stale nodes and swap them out for more requested regions, in case we breach the threshold due to overpopulation.

In case of an overflow, we utilize the **importance** attribute of a node in conjunction with the product of the number of accesses to a node (updated every time it gets accessed), and a time decay function that takes into consideration the last time the node was accessed. Using this metric, which we call `adjusted node imporance`, our cache pruning strategy takes into consideration both the relevance of a node at a particular instant alongside the contents of the sub-tree. Sub-trees in ARGUS are replaced based on this adjusted importance score. To leverage the spatial and temporal locality of access patterns, when a request for a spatiotemporal region comes in, we mark

both the set of Cells in that region and the immediate spatiotemporal neighborhood of that region as being of future interest as prescribed in [11].

Chapter 10

Building Multiple Regional Models At Scale

Building an all-encompassing singular model for a large spatial extent (we consider the continental US or CONUS in this study) is infeasible due to the accompanying resource requirements. Such a model would entail an exceptionally large number of parameters; the resource overheads for training would be prohibitive. Distributed training is feasible; however, these tend to incur high network costs as the parameter weight vectors are reconciled. Complex phenomena, such as climate, captured in these geospatial datasets are typically subject to variability at the local scales , which affect the accuracy of a singular global model. In such cases, training models for smaller spatial extents (or local learning [123]) may be a more reasonable approach.

Our framework, **RELAY**, attempts to combine the effectiveness of both global and localized models. To preserve accuracy, we build a multiplicity of models, each tuned to its particular spatial extent, that capture subtle regional variations. To facilitate this, we partition the CONUS into N smaller spatial extents that are contiguous and non-overlapping. Our partitioning scheme is deterministic, and the size of the spatial extent for which the models are built is based on the complexity of the network and resolution of the data. During model training, we need to account for the residency of not just the input feature values but also the intermediate tensors that are created within the network. Our objective is to effectively train N models each tuned to the particular spatial extent.

The key enabling idea in our methodology is the novel use of transfer learning schemes to train, refine, and ensure effective starting weights for deep networks. Rather than train each constituent model instance from scratch (cold-start), we ensure that model instances have a superior starting point (warm-start) for their weight vectors and coefficients for faster convergence.

We design and explore three different transfer learning schemes: amalgamated, partitioned, and hybrid. Transfer learning has been used primarily in cases where layers/network trained over a given dataset are reused within networks trained over a different dataset. We transfer learn across

models built for different spatial locations of the same dataset. This is different from traditional transfer learning which occurs across models purpose-built for different objectives over different datasets. We use transfer learning to calibrate the layers that can be used as is and those that need to be retrained but with effective starting points. The earlier parts of network are tagged as fixed and subsequent layers trained from the specified starting points.

We first partition the geographical area under consideration (CONUS for this study) into fixed sized spatial extents. Our methodology relies on rigorous and extensive training of a minimal number of models and then preforming a targeted diffusion of those model characteristics to relevant spatial extents. We designed three different transfer learning schemes: amalgamated, partitioned, and hybrid transfer learning schemes. In our transfer learning schemes we train one or more anchor model instances rigorously. Once trained, several characteristics of these anchor models including weight vectors, coefficients, and model hyperparameters are diffused in a targeted fashion to other spatial extents as part of the transfer learning process. During rigorous training of the anchor models, we also allow users to experiment with the accuracy measures such as RMSE, MAE, or SSIM that are most suited for the task at hand. Once the desired accuracy measure has been settled on, it is used to enforce stopping criteria for model instances being constructed for the spatial extent. The stopping criteria for the individual model instances is based on achieving 95% of accuracy of the rigorously trained anchor models whose characteristics are being diffused as part of the transfer learning process.

To conserve resources, each of the transfer learned models are trained with data locality (avoids network I/O), with data grouped into batches (to reduce memory residency requirements), and leverages progressive sampling to gradually increase the amount of data fed to the model until the desired accuracy is achieved. More importantly, because we use transfer learning, earlier parts of the network are fixed (i.e., they are non-trainable) while the latter parts are trained and tuned to account for subtle regional variations. Cumulatively, the use of transfer learning alleviates the vanishing gradients problem and avoids duplicate processing costs that are incurred if each model has a cold-start and must be trained from scratch.

In the amalgamated transfer learning scheme, we construct a global anchor model for the complete hypothetical spatial extent. The training data for this phase is drawn from the entire CONUS and we uniformly sample $n\%$ ($n << 100$) of data from all spatial extents. This facilitates spatial coverage by ensuring that data from all spatial extents are represented in the training datasets. This also allows the anchor model to capture broad variations in the feature space that occur across the CONUS. This anchor model is rigorously trained and includes hyperparameter tuning. The weight vectors, coefficients, and hyperparameters are then diffused to the N spatial extents as the starting points and the models trained with progressive sampling and the relevant stopping criteria (95% of the anchor model accuracy). Since the anchor model is trained with data from all spatial extents, individual model instances (that warm start using the anchor model) are robust to data outliers.

In the partitioned transfer learning scheme, the N spatial extents are partitioned into K disjoint subsets based on spatial similarity. For each disjoint subset we train one anchor model rigorously. The anchor models are chosen based on their proximity to the cluster centroid. We posit that the weights diffused from the anchor model to other spatially similar extents are superior because the data distribution characteristics within spatially similar regions are likely to be similar.

The hybrid transfer learning scheme combines amalgamated and partitioned learning. Rather than train the anchor models within each cluster from scratch, we use weight vectors from a global anchor model (like in amalgamated mode) as the starting point - this allows the centroid models to converge faster. The weights diffused from the centroid model include those from the anchor models (initial layers) and the centroid model (intermediate and final layers). When training model instances the weights for the initial and intermediate layers are held fixed, while the final layer weights are trainable and use the anchor model weights as the starting point.

Our methodology facilitates training a collection of deep learning models, each tuned to their particular spatial extent. To ensure efficient training of these models while reducing the resource requirements, we include data partitioning schemes to ensure dispersion and data locality during model training. Next, we describe the design of our transfer learning schemes to reduce cold-start overheads for model training. By facilitating effective reuse of layers of the deep network,

we minimize duplicate processing overheads and reduce resource requirements. We describe our transfer learning schemes (amalgamated, partitioned, and hybrid) and the rationale for their design. We leverage these schemes to inform the degree of transfer learning to calibrate the degree and layers of the network to be reused.

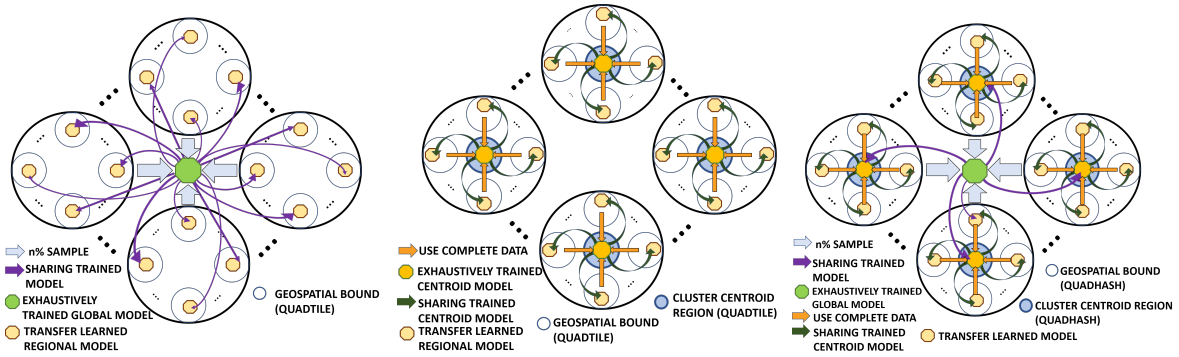
10.1 Data partitioning

We ensure that the unit of spatial data-partitioning can also be used to express the spatial bounds for each regional model. Our spatial data partitioning scheme relies on quadkeys generated from quadtiles [121]. The advantages of using quadkeys are three-fold. First, the quadkey's length allows us to control the size of the spatial extent. Second, the scheme is deterministic in the sense that a given quadkey uniquely and unambiguously identifies a particular spatial extent. Finally, both the identification of the spatial extent from a quadkey and the partitioning a much larger spatial extent into smaller extents can be performed in a hierarchical and decentralized fashion.

Once the precision of a quadtile is finalized, we use it to partition data along the hash boundaries. Our server pool is organized as a DHT (distributed hash table). DHTs organize data as <key,value> pairs. Typically, the key is generated by computing the hash value of the content. The hash space is partitioned among the server nodes and each node is responsible for a contiguous portion of the hash space. RELAY uses quadkey as the input to the hashing function, which then disperses content to the relevant server node. Using the quadkey allows us to ensure that all data for a particular spatial extent will be stored on the same server node. Each server node is also responsible for multiple quadkey codes. This co-location ensures data locality during training and minimizes the network I/O during training. Finally, identifying the server node where data is stored is performed in a deterministic and decentralized fashion.

We use the zero-hop DHT refinement that allows maintaining hash space partitioning information, i.e., for a collection of servers the system maintains the key ranges. With a server pool of N nodes, the computational cost to locate a data-region is $O(1)$. Distribution properties of the

hashing function ensures that data from different spatial extents are load-balanced and that there are no skews in data storage within the cluster.



(a) Amalgamated Approach **(b)** Partitioned Approach **(c)** Hybrid Approach

Figure (10.1) Types of Distributed Transfer Learning Implementations

10.2 Partitioning of Geospatial Domain

The crux of the RELAY framework is to train a small number of regional models exhaustively via cold-start and reuse the learned weights of these models to train the remainder regional models through transfer learning. Spatiotemporal datasets, such as satellite imagery data, often have regional variations. We aim to group regions with similar characteristics to identify a representative (**anchor**) region and the remaining **dependent** regions for each group.

Our transfer learning approach utilizes two possible grouping schemes for spatiotemporal regions: one based on geospatial proximity and the other based on metadata. The **proximity-based** scheme builds on the hypothesis that in geospatial datasets, data in proximate spatiotemporal bounds have similar characteristics. This also dovetails with the rationale that models built over proximate spatiotemporal regions can be trained via transfer learning from one another (i.e. faster convergence with similar accuracy). The **metadata-based** grouping scheme is a refinement of our methodology that utilizes auxiliary metadata associated with spatial extents, if available, for clustering similar regions. We utilize the National Land Cover Database (NLCD) codes [102] to help group regions with similar land/ topographical characteristics. In this scheme, we use the land type

of each pixel within a quadtile bound to form a 21-dimensional vector (one for each NLCD land type) that represents the fraction of each land-cover type in the quadhash region. The representative vector (one for each quadtile bound) is used to cluster these regions using k-means++ [124].

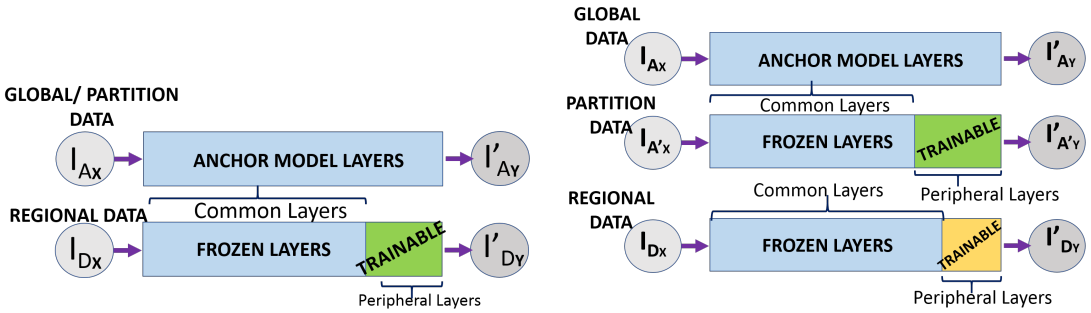
The proximity-based grouping ensures co-location between the dependent and the anchor models, since our underlying dataset is partitioned based on quadtile strings. Metadata-based clustering, along with cluster optimization is a one-time process that needs to happen before actual training can begin. Metadata-based clustering can result in more accurate clustering of regions leading to a significantly smaller number of clusters. This would lead to fewer exhaustively-trained anchor models and potentially higher throughput. When auxiliary metadata is unavailable a generalized proximity-based approach is more suitable.

10.3 Rigorous vs Assisted Training of Models

Our methodology allows users to experiment with the structure of the network layers, loss functions, regularization schemes etc. Additionally, users can tune hyperparameters such as learning rates that are deeply aligned with the network structure and the data characteristics for better model optimization. However, hyper-parameter optimization can be an expensive operation, especially given the number of regional models involved. We perform extensive modeling on a much smaller subset of models, which we call **anchor models**, and use these rigorously-trained models as the basis to train their corresponding dependent models built over similar spatiotemporal regions.

In our transfer learning schemes, we also experiment with the *Degree of Transfer Learning*. We explore how many layers weights need to be transferred, i.e., we freeze the weights of a (initial) portion of the deep layers of the anchor models and re-train a small fraction of the latter layers over regional data to generate the dependent models. By controlling the degree of transfer learning within a spatially similar cluster, we aim to reduce both resource utilization for training large models as well as increase their convergence rate.

Degree Of Transfer Learning: This signifies the amount of network layers for a dependent model that we keep frozen during the transfer learning phase. In our approach, this degree of transfer



(a) Amalgamated/ Partitioned **(b)** Hybrid

Figure (10.2) Model Layers For Different Transfer Learning Schemes

learning is directly proportional to the similarity between the spatiotemporal extents represented by the anchor and dependent models, as shown in Fig. 10.2.

The portion of anchor model layers that are frozen during the training of dependent models depends on the problem at hand. We explain the degrees of transfer learning for each of our individual approaches below.

Transfer Learning Phases: In the first phase of transfer learning, we implement *feature extraction* by freezing all the layers of an anchor model except the final few fully-connected layers of a model and training over a portion ($s\%$) of the overall regional data. The remaining portion of the anchor model is integrated directly into the new neural network model and not updated while training the dependent model.

Upon convergence of this phase or if the model accuracy is within 95% of the corresponding anchor model, in an optional phase, we subsequently unfreeze the entire dependent model and train it using regional data over a smaller number of epochs over a smaller learning rate to fine-tune the internal neural network layers to the new domain.

10.4 The Amalgamated Transfer Learning Scheme

In the amalgamated transfer learning scheme (Fig.10.1a), we train a single anchor model and use it to bootstrap training all other regional models over our data space. This is suitable in cases where the data has relatively low regional variations. Consequently, a single global anchor model should be able to learn the overall characteristics of the feature space including non-linear inter-

actions; this can be re-purposed to fine-tune the dependent models with regional information. The global anchor model is trained rigorously over $n\%(n << 100)$ of the overall data in a distributed manner. For this, we leverage Pytorch Lightning [119] in Distributed Data Parallel (DDP) mode.

Using this global anchor model over the entire CONUS, we partially retrain the weights learned and reuse structure of the network, to transfer learn across the N regional spatial extents. In our implementation, N is determined by the number of data regions of quadkey length 11 that comprise our data space.

Degree of Transfer Learning: Since we use a single global model as an anchor to train all other regional models, we assume the spatiotemporal domains represented by the global and the dependent models might be subject to variations. We allow for a lesser degree of transfer learning between the global anchor model and dependent regional models.

Fig. 10.2a depicts the degree of transfer learning for both amalgamated and partitioned transfer learning modes. In case of amalgamated, due to transfer learning between a global and regional domains, we reduce the number of layers of the anchor/parent model frozen during training of the dependent/child models. This leads to relatively slower convergence rate, but better model accuracy.

10.5 The Partitioned Transfer Learning Scheme

Our partitioned transfer learning scheme is more suited for datasets where data domains have strong spatiotemporal correlations, i.e., data from similar regions have similar characteristics. We cluster similar spatial extents into K groups and perform transfer learning within each such group (partition). We train K exhaustive anchor models, one for each group, and train other models within a cluster based on their anchors.

Our segmentation groups spatially similar regions to form K non-overlapping partitions. For each such partition, we identify a spatiotemporal region as a centroid. We identify the centroid region as one that is equidistant from the other regions in this group. Since we deal with spatiotemporal datasets and create partition based on quadkeys, we select the quadtile in the group

that is closest to the central quadtile as the region for which the anchor model is to be built.

Degree of Transfer Learning: Since anchor regions in the same spatiotemporal partition are relatively similar to their dependent regions, dependent models in partitioned scheme have a higher degree of transfer learning, i.e., fewer peripheral neural network layers are unfrozen during transfer learning (Fig. 10.2a). This should lead to faster convergence rate for the dependent models and relatively good accuracy in cases where the variance in data domain have the desired characteristics mentioned above.

10.6 The Hybrid Transfer Learning Scheme

The hybrid transfer learning scheme (Fig. 10.2b) combines aspects of both Amalgamated and Partitioned transfer learning schemes and is suitable for scenarios where the overall data domain has characteristics that are a mix between the two scenarios mentioned before. In this implementation, we have 2 sets of anchor models. The primary anchor model is a global model built similar to the amalgamated phase utilizing $n\%$ of the data. Secondary anchor models are built for the K partition centroids through transfer learning using the primary anchor model in a rigorous fashion. The spatial partitioning scheme and centroid assignment is similar to the description in Section 10.5. The remainder of the dependent models are trained similar to the partitioned transfer learning scheme using their corresponding secondary anchor model as the base.

In our hybrid scheme, we expect the overall throughput to be lower than the previous implementations due to the 2 stages of exhaustive modelling before the training of the dependent models, but the hybrid scheme can have potentially higher accuracy in cases where data characteristics do not fully fit the mould of the previous two implementations.

Chapter 11

Conclusions

Through our research, we aim to facilitate interactive spatiotemporal data visualizations and analytics. We address challenges affecting latency of such evaluations over large scale datasets relating to excessive/unnecessary disk and network I/O between the clients and the distributed server, excessive record processing and analytics over high-dimensional data objects.

RQ1: How can we ensure that front-end users are not overwhelmed by the scope of the underlying data?

We implement a unified query scheme understandable by both the in-memory distributed cache framework as well as the distributed data indexing/partitioning scheme. This ensures that the front-end interfaces are kept simple, with their only major task being translation of user actions into spatiotemporal queries and vice-versa.

RQ2: How can we ensure interactivity of spatiotemporal query evaluation over voluminous datasets?

Our distributed, dynamic caching scheme (STASH) alleviates I/O overheads associated with disk accesses. We also facilitate rapid data discovery that minimizes query-forwarding between the cluster nodes and local traversals within the data structure. STASH's maintenance scheme is sensitive to spatial and temporal locality of users' access patterns that help increase its hit-rate. We also utilize a client-side, stand-alone version on STASH that precludes excessive data retrievals and transfers from frequent view changes triggered by a user and a space-efficient super-resolution model (Glance), that works in conjunction and accurately upscales low-dimensional data through multiple upscale factors over multiple levels of resolution.

RQ3: How can our system cope with skews in access patterns (that may lead to hotspots)

while preserving latencies and good perceptual quality?

We enable STASH with a dynamic, decentralized replication scheme that helps alleviate frequent hotspots that is common in dominant geospatial explorations by replicating highly accessed data fragments in nodes whose data domain is diagonally opposite to the hotspotted regions.

RQ4: How can we preserve accuracy of our query results and make them adaptive to a fast-evolving data-store?

We couple the both GLANCE and ARGUS models with an error-mitigation module that can identify potential error-prone regions in the data-space to help inform regions of low-fidelity. This module can also trigger re-training if errors go below a pre-defined threshold. Additionally in case of RUBIKS, we implement a Welford's online algorithm for maintaining running aggregate statistics which are fully accurate. In case of correlation measures, we implement a weighted kernel approach for computing covariance between non-concurrent measurements.

RQ5: How can we extract meaningful insights, capable of being rendered on a 2D visual interface, from a multi-dimensional data-store?

Finally, we design a set of memory-resident deep neural network-based models (GLANCENET and ARGUSNET) for extraction of insights from high-dimensional data (multi-spectral imagery) objects during query evaluation over a distributed cluster. We ensure collaborative transfer of knowledge between these models and robust training through generalization by training these models through Multi-Task Learning.

Bibliography

[1] Elena Geanina Ularu, Florina Camelia Puican, Anca Apostu, Manole Velicanu, et al. Perspectives on big data and big data analytics. *Database Systems Journal*, 3(4):3–14, 2012.

[2] Mackinlay Card. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[3] Pat Hanrahan. Analytic database technologies for a new kind of user: the data enthusiast. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 577–578, 2012.

[4] Jeffrey Heer and Ben Shneiderman. Interactive dynamics for visual analysis: A taxonomy of tools that support the fluent and flexible use of visualizations. *Queue*, 10(2):30–55, 2012.

[5] Zoltán Konyha, Krešimir Matkovic, and Helwig Hauser. Interactive visual analysis in engineering: A survey. *Posters at SCCG*, 2009:31–38, 2009.

[6] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile networks and applications*, 19(2):171–209, 2014.

[7] Jae-Gil Lee and Minseo Kang. Geospatial big data: challenges and opportunities. *Big Data Research*, 2(2):74–81, 2015.

[8] *Powers of 10: Time Scales in User Experience*, 2019. https://www.nngroup.com/articles/powers-of-10-time-scales-in-ux/.

[9] Danyel Fisher. Hotmap: Looking at geographic attention. *IEEE transactions on visualization and computer graphics*, 13(6):1184–1191, 2007.

[10] Lei Shi, Zhimin Gu, Lin Wei, and Yun Shi. Quantitative analysis of zipf's law on web cache. In *International Symposium on Parallel and Distributed Processing and Applications*, pages 845–852. Springer, 2005.

[11] Saptashwa Mitra, Paahuni Khandelwal, Shrideep Pallickara, and Sangmi Lee Pallickara. Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations. In *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–11. IEEE, 2019.

[12] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[13] Saptashwa Mitra, Daniel Rammer, Shrideep Pallickara, and Sangmi Lee Pallickara. Glance: A generative approach to interactive visualization of voluminous satellite imagery. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 359–367. IEEE, 2021.

[14] Saptashwa Mitra, Maxwell Roselius, Pedro Andrade-Sanchez, John K McKay, and Sangmi Lee Pallickara. Radix+: High-throughput georeferencing and data ingestion over voluminous and fast-evolving phenotyping sensor data. *Concurrency and Computation: Practice and Experience*, 35(8):e7484, 2023.

[15] Saptashwa Mitra, Daniel Rammer, Shrideep Pallickara, and Sangmi Lee Pallickara. A generative approach to visualizing satellite data. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 815–816. IEEE, 2021.

[16] Saptashwa Mitra, Paahuni Khandelwal, Shrideep Pallickara, and Sangmi Pallickara. Argus: Rapid tracking of wildfires from unlabeled satellite images. In *International Conference on Cloud Computing (CLOUD)*, 2023.

[17] Abdul Matin, Samuel Armstrong, Saptashwa Mitra, Shrideep Pallickara, and Sangmi Lee Pallickara. Rapid betweenness centrality estimates for transportation networks using capsule networks. In *2022 Fourth International Conference on Transdisciplinary AI (TransAI)*, pages 89–96. IEEE, 2022.

[18] Saptashwa Mitra, Menuka Warushavithana, Mazdak Arabi, Jay Breidt, Sangmi Pallickara, and Shrideep Pallickara. Alleviating resource requirements for spatial deep learning work-

loads. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 452–462. IEEE, 2022.

[19] Caleb Carlson, Menuka Warushavithana, Saptashwa Mitra, Kassidy Barram, Sudipto Ghosh, Jay Breidt, Sangmi Lee Pallickara, and Shrideep Pallickara. Resource efficient profiling of spatial variability in performance of regression models. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 437–444. IEEE, 2022.

[20] Menuka Warushavithana, Kassidy Barram, Caleb Carlson, Saptashwa Mitra, Sudipto Ghosh, Jay Breidt, Sangmi Lee Pallickara, and Shrideep Pallickara. A framework for profiling spatial variability in the performance of classification models. *Under Review*.

[21] Menuka Warushavithana, Saptashwa Mitra, Mazdak Arabi, Jay Breidt, Sangmi Lee Pallickara, and Shrideep Pallickara. Containerization of model fitting workloads over spatial datasets. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 3770–3779. IEEE, 2021.

[22] Menuka Warushavithana, Caleb Carlson, Saptashwa Mitra, Daniel Rammer, Mazdak Arabi, Jay Breidt, Sangmi Lee Pallickara, and Shrideep Pallickara. Distributed orchestration of regression models over administrative boundaries. In *2021 IEEE/ACM 8th International Conference on Big Data Computing, Applications and Technologies (BDCAT'21)*, pages 80–90, 2021.

[23] Menuka Warushavithana, Saptashwa Mitra, Mazdak Arabi, Jay Breidt, Sangmi Lee Pallickara, and Shrideep Pallickara. A transfer learning scheme for time series forecasting using facebook prophet. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 809–810. IEEE, 2021.

[24] Xiaochuang Yao and Guoqing Li. Big spatial vector data management: a review. *Big Earth Data*, 2(1):108–129, 2018.

[25] The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[26] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[27] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep 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. IEEE, 2011.

[28] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Exploiting geospatial and chronological characteristics in data streams to enable efficient storage and retrievals. *Future Generation Computer Systems. Elsevier.*, 29(4):1049–1061, 2013.

[29] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Polygon-based query evaluation over geospatial data using distributed hash tables. In *IEEE/ACM Conference on Utility and Cloud Computing.*

[30] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Geometry and proximity constrained query evaluations over large geospatial datasets using distributed hash tables. *IEEE Computing in Science and Engineering (CiSE). Special Issue on Extreme Data*, 16(4):53–60, 2014.

[31] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, 5(1):28–42, 2017.

[32] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Analytic queries over geospatial time-series data using distributed hash tables. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1408–1422, 2016.

[33] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Expressive query support for multidimensional data in distributed hash tables. In *Proceedings of the IEEE/ACM Conference on Utility and Cloud Computing*.

[34] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Hermes: Federating fog and cloud nodes to support query evaluations in continuous sensing environments. *IEEE Cloud Computing. Special Issue on Autonomic Clouds.*, 4(2):54–62, 2017.

[35] Wes Lloyd, Shrideep Pallickara, Olaf David, Mazdak Arabi, Tyler Wible, Jeffrey Ditty, and Ken Rojas. Demystifying the clouds: Harnessing resource utilization models for cost effective infrastructure alternatives. *IEEE Transactions on Cloud Computing.*, 5(4):667–680, 2017.

[36] Wes Lloyd, Shrideep Pallickara, Olaf David, Jim Lyon, Mazdak Arabi, and Ken Rojas. Migration of multi-tier applications to infrastructure-as-a-service clouds: An investigation using kernel-based virtual machines. In *12th IEEE/ACM International Conference on Grid Computing*, pages 137–144.

[37] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 15–28, 2012.

[38] Leilani Battle, Remco Chang, and Michael Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1363–1375. ACM, 2016.

[39] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[40] Lauro Lins, James T Klosowski, and Carlos Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2456–2465, 2013.

[41] Cicero AL Pahins, Sean A Stephens, Carlos Scheidegger, and Joao LD Comba. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE transactions on visualization and computer graphics*, 23(1):671–680, 2017.

[42] Wenbo Tao, Xiaoyu Liu, Çagatay Demiralp, Remco Chang, and Michael Stonebraker. Kyrix: Interactive visual data exploration at scale. CIDR, 2019.

[43] Luís Santos, João Coutinho-Rodrigues, and Carlos Henggeler Antunes. A web spatial decision support system for vehicle routing using google maps. *Decision Support Systems*, 51(1):1–9, 2011.

[44] Richard Wesley, Matthew Eldridge, and Pawel T Terlecki. An analytic data engine for visualization in tableau. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1185–1194. ACM, 2011.

[45] Nivan Ferreira, Jorge Poco, Huy T Vo, Juliana Freire, and Cláudio T Silva. Visual exploration of big spatio-temporal urban data: A study of new york city taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013.

[46] Michael Stonebraker, Paul Brown, Donghui Zhang, and Jacek Becla. Scidb: A database management system for applications with complex analytics. *Computing in Science & Engineering*, 15(3):54, 2013.

[47] Paul Ramsey et al. Postgis manual. *Refractions Research Inc*, 17, 2005.

[48] Cliff Engle, Antonio Lupher, Reynold Xin, Matei Zaharia, Michael J Franklin, Scott Shenker, and Ion Stoica. Shark: fast data analysis using coarse-grained distributed memory. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 689–692. ACM, 2012.

[49] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[50] Rui Li, Yinfeng Zhang, Zhengquan Xu, and Huayi Wu. A load-balancing method for network giss in a heterogeneous cluster-based system using access density. *Future Generation Computer Systems*, 29(2):528–535, 2013.

[51] Rui Li, Wei Feng, Huayi Wu, and Qunying Huang. A replication strategy for a distributed high-speed caching system based on spatiotemporal access patterns of geospatial data. *Computers, Environment and Urban Systems*, 61:163–171, 2017.

[52] Sanjoy Paul and Zongming Fei. Distributed caching with centralized control. *Computer Communications*, 24(2):256–268, 2001.

[53] Rui Li, Jiapei Fan, Xinxing Wang, Zhen Zhou, and Huayi Wu. Distributed cache replacement method for geospatial data using spatiotemporal locality-based sequence. *Geo-spatial Information Science*, 18(4):171–182, 2015.

[54] Shaoming Pan, Lian Xiong, Zhengquan Xu, Yanwen Chong, and Qingxiang Meng. A dynamic replication management strategy in distributed gis. *Computers & geosciences*, 112:1–8, 2018.

[55] *Tableau Desktop*, 2019. https://www.tableau.com/products/desktop.

[56] Miguel García and Barry Harmsen. *Qlikview 11 for developers*. Packt Publishing Ltd, 2012.

[57] Alex Berson and Stephen J Smith. *Data warehousing, data mining, and OLAP*. McGraw-Hill, Inc., 1997.

[58] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *The craft of information visualization*, pages 364–371. Elsevier, 2003.

[59] Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[60] Tile layers, April 2020.

[61] Tiled web map, December 2019.

[62] 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.

[63] GC Fox, Hasan Bulut, Kangseok Kim, Sung-Hoon Ko, Sangmi Lee, Sangyoon Oh, Shrideep Pallickara, Xiaohong Qiu, Ahmet Uyar, and Minjun Wang. Collaborative web services and peer-to-peer grids. *SIMULATION SERIES*, 35(1):3–12, 2003.

[64] Sangmi Lee, Geoffrey Fox, Sunghoon Ko, Minjun Wang, and Xiaohong Qiu. Ubiquitous access for collaborative information system using svg. In *Proceedings of SVGopen conference July*.

[65] Shrideep Pallickara and Geoffrey C Fox. On the matching of events in distributed brokering systems. In *ITCC (2)*, pages 68–76.

[66] Shrideep Pallickara and Geoffrey Fox. Naradabrokering: a distributed middleware framework and architecture for enabling durable peer-to-peer grids. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 41–61. Springer.

[67] Geoffrey Fox, Galip Aydin, Hasan Bulut, Harshawardhan Gadgil, Shrideep Pallickara, Marlon Pierce, and Wenjun Wu. Management of real-time streaming data grid services. *Concurrency and Computation: Practice and Experience*, 19(7):983–998, 2007.

[68] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[69] Yifan Wang, Federico Perazzi, Brian McWilliams, Alexander Sorkine-Hornung, Olga Sorkine-Hornung, and Christopher Schroers. A fully progressive approach to single-image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 864–873, 2018.

[70] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[71] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

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

[73] Kui Jiang, Zhongyuan Wang, Peng Yi, Junjun Jiang, Jing Xiao, and Yuan Yao. Deep distillation recursive network for remote sensing imagery super-resolution. *Remote Sensing*, 10(11):1700, 2018.

[74] Mehdi SM Sajjadi, Bernhard Scholkopf, and Michael Hirsch. Enhancenet: Single image super-resolution through automated texture synthesis. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4491–4500, 2017.

[75] Kui Jiang, Zhongyuan Wang, Peng Yi, and Junjun Jiang. A progressively enhanced network for video satellite imagery superresolution. *IEEE Signal Processing Letters*, 25(11):1630–1634, 2018.

[76] Jinshan Pan, Sifei Liu, Deqing Sun, Jiawei Zhang, Yang Liu, Jimmy Ren, Zechao Li, Jinhui Tang, Huchuan Lu, Yu-Wing Tai, et al. Learning dual convolutional neural networks for low-level vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3070–3079, 2018.

[77] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[78] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[79] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

[80] Barret Zoph, Deniz Yuret, Jonathan May, and Kevin Knight. Transfer learning for low-resource neural machine translation. *arXiv preprint arXiv:1604.02201*, 2016.

[81] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

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

[83] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[84] *Grafana Labs*, 2019. https://grafana.com/grafana.

[85] *React*, 2023. https://react.dev/.

[86] *Welcome to Flask — Flask Documentation (2.3.x)*, 2023. https://flask.palletsprojects.com/en/2.3.x/.

[87] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on networking*, 11(1):17–32, 2003.

[88] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Galileo: A framework for distributed storage of high-throughput data streams. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 17–24. IEEE, 2011.

[89] G. Niemeyer. *Geohash*, 1999. http://www.geohash.org/.

[90] *Elastic Search*, 2019. https://www.elastic.co/guide/en/elasticsearch/reference/6.2/index.html.

[91] Chaowei Yang, Michael Goodchild, Qunying Huang, Doug Nebert, Robert Raskin, Yan Xu, Myra Bambacus, and Daniel Fay. Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? *International Journal of Digital Earth*, 4(4):305–329, 2011.

[92] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 34. ACM, 2008.

[93] *National Oceanic and Atmospheric Administration, The North American Mesoscale Forecast System*, 2019. http://www.emc.ncep.noaa.gov/index.php?branch=NAM.

[94] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1:29–53, 1997.

[95] Guoren Wang, Yue Zeng, Rong-Hua Li, Hongchao Qin, Xuanhua Shi, Yubin Xia, Xuequn Shang, and Liang Hong. Temporal graph cube. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[96] Saptashwa Mitra, Matt Young, Sangmi Lee Pallickara, and Shrideep Pallickara. Rubiks: Rapid explorations and summarization over high dimensional spatiotemporal datasets. *Under Review*.

[97] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[98] Walter Hugo Lopez Pinaya, Sandra Vieira, Rafael Garcia-Dias, and Andrea Mechelli. Autoencoders. In *Machine learning*, pages 193–208. Elsevier, 2020.

[99] Paul R Seaber, F Paul Kapinos, and George L Knapp. *Hydrologic unit maps*, volume 2294. US Government Printing Office Washington, DC, USA, 1987.

[100] Kira Rehfeld, Norbert Marwan, Jobst Heitzig, and Jürgen Kurths. Comparison of correlation analysis techniques for irregularly sampled time series. *Nonlinear Processes in Geophysics*, 18(3):389–404, 2011.

[101] Ugur Demir and Gozde Unal. Patch-based image inpainting with generative adversarial networks. *arXiv preprint arXiv:1803.07422*, 2018.

[102] Collin Homer, Jon Dewitz, Limin Yang, Suming Jin, Patrick Danielson, George Xian, John Coulston, Nathaniel Herold, James Wickham, and Kevin Megown. Completion of

the 2011 national land cover database for the conterminous united states–representing a decade of land cover change information. *Photogrammetric Engineering & Remote Sensing*, 81(5):345–354, 2015.

[103] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[104] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[105] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[106] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[107] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[108] Xiaojiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. *Advances in neural information processing systems*, 29:2802–2810, 2016.

[109] Sangmi Lee Pallickara, Shrideep Pallickara, and Marlon Pierce. Scientific data management in the cloud: A survey of technologies, approaches and challenges. *Handbook of Cloud Computing*, pages 517–533, 2010.

[110] Saptashwa Mitra, Yu Qiu, Haley Moss, Kaigang Li, and Sangmi Lee Pallickara. Effective integration of geotagged, ancilliary longitudinal survey datasets to improve adulthood obesity predictive models. In *2018 17th IEEE International Conference On Trust, Security And*

Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pages 1738–1746. IEEE, 2018.

[111] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, 5(1):28–42, 2015.

[112] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Hermes: Federating fog and cloud domains to support query evaluations in continuous sensing environments. *IEEE Cloud Computing*, 4(2):54–62, 2017.

[113] Walid Budgaga, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. A framework for scalable real-time anomaly detection over voluminous, geospatial data streams. *Concurrency and Computation: Practice and Experience*, 29(12):e4106, 2017.

[114] Esri. *An overview of the Spatial Analyst toolbox*. https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-analyst/an-overview-of-the-spatial-analyst-toolbox.htm.

[115] Wilfrid Schroeder, Patricia Oliva, Louis Giglio, and Ivan A Csiszar. The new viirs 375 m active fire detection data product: Algorithm description and initial assessment. *Remote Sensing of Environment*, 143:85–96, 2014.

[116] Fire perimeters, March 2023.

[117] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

[118] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. Ieee, 2016.

[119] WA Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[120] Anuj Karpatne, Gowtham Atluri, James H Faghmous, Michael Steinbach, Arindam Banerjee, Auroop Ganguly, Shashi Shekhar, Nagiza Samatova, and Vipin Kumar. Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on knowledge and data engineering*, 29(10):2318–2331, 2017.

[121] Quadtiles, November 2018.

[122] Marko Luksa. *Kubernetes in action*. Simon and Schuster, 2017.

[123] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning. *Lazy learning*, pages 11–73, 1997.

[124] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

Appendix A

License

Colorado State University LaTeX Thesis Template

by Saptashwa Mitra – 2023

This is free and unencumbered software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

In jurisdictions that recognize copyright laws, the author or authors of this software dedicate any and all copyright interest in the software to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.