Enabling Fast, Effective Visualization of Voluminous Gridded Spatial Datasets

Paahuni Khandelwal, Menuka Warushavithana, Sangmi Lee Pallickara, Shrideep Pallickara
Department of Computer Science
Colorado State University
Fort Collins, Colorado, 80521
paahuni@colostate.edu, menukaw@colostate.edu, sangmi.pallickara@colostate.edu, shrideep.pallickara@colostate.edu

***Abstract*—Gridded spatial datasets arise naturally in environmental, climatic, meteorological, and ecological settings. Each grid point encapsulates a vector of variables representing different measures of interest. Gridded datasets tend to be voluminous since they encapsulate observations for long timescales. Visualizing such datasets poses significant challenges stemming from the need to preserve interactivity, manage I/O overheads, and cope with data volumes. Here we present our methodology to significantly alleviate I/O requirements by leveraging deep neural network-based models and a distributed, in-memory cache to facilitate interactive visualizations. Our benchmarks demonstrate that deploying our lightweight models coupled with back-end caching and prefetching schemes can reduce the client's query response time by 92.3% while maintaining a high perceptual quality with a PSNR (peak signal-to-noise ratio) of 38.7 dB.**

***Index Terms*—gridded datasets, interactive visualizations, spatial transfer learning, distributed cache, geo-ai**

I. Introduction

The proliferation of sensors, observational equipment, and networked measurement devices has contributed to substantial growth in data volumes. In addition to the observations of interest, spatial datasets have geocodes (such as $\langle lat/long \rangle$), associated with them. Several data also have timestamps associated with the observations; the observations are often multidimensional encapsulating multiple, related features of interest. We consider gridded spatial datasets that arise frequently in climate, meteorological, ecological, and satellite-based remote sensing datasets. In gridded spatial datasets, the data are available at fixed, spatially dispersed points based on the spatial resolution which determines the spatial increments at which data are available. For e.g., a 1 km spatial resolution represents the case that data are available at 1 km spatial increments. Gridded spatial datasets also arise in statistically downscaled datasets that produce data points using interpolation techniques.

Gridded spatial datasets tend to be voluminous, and the data vector associated with individual grid points includes the spatial coordinates and chronological timestamps associated with them. These datasets may be managed using relational, No-SQL, or document-oriented storage formats. Observations at each grid point for a particular timestamp are organized as a single record. Retrieving data for a viewport, thus entails retrievals of records corresponding to the spatial extent encapsulating the viewport.

Scientists, stakeholders, and users alike rely on visualizations to understand spatial variation of phenomena. The pairwise combinations in which they can be layered have the asymptotic bound of $O(N^2)$ where N is the number of variables. Each feature of interest may be visualized independently, or multiple features may be layered to understand how they vary with respect to each other. The crux of this effort is to enable interactive visualizations of gridded spatial datasets.

A. *Challenges*

Visualizing gridded spatial datasets introduces challenges that stem from:

1) **I/O costs**: Visualizations entail data retrievals from the server side entailing both disk I/O and network I/O. A related issue is that the back-end storage subsystem services multiple users concurrently. As an increasing number of requests come in, I/O contention increases causing throughputs to plummet on the server side.
2) **Interactivity**: During visualization operations, users expect timeliness of the rendering operations during pivots, panning, roll-ups, and drill-downs.
3) **Data volumes**: Besides individual data points being multidimensional, the data are voluminous and exacerbate challenges relating to both I/O and interactivity.

B. *Research Questions*

The overall objective of this effort is to significantly reduce I/O operations performed during ST visualizations. Within this broader goal, we explore the following research questions:

RQ-1: How can we effectively design models that render multivariate phenomena over large spatial extents?

RQ-2: How can we alleviate the resource-intensive nature of model training for large spatiotemporal extents?

RQ-3: How can we effectively combine the space-efficiency of models with the information available in voluminous, ground-truth observational data?

C. *Approach Summary*

Our methodology targets interactive visualizations of gridded spatial datasets. We get there in four phases: (1) we partition the viewport into tiles and use models to render tile visualizations; (2) we design and train models that generate effective visualizations; (3) manage the computational complexity of training models; and (4) design a scalable, distributed cache to seed models during inferences.

We first partition the viewport into tiles. The size of the tiles (i.e., the spatial extent that they represent) is dependent on the zoom level. Similarly, the number of titles that comprise the viewport is also dependent on the zoom levels.

Rather than extensively performing disk and network I/O during visualizations, we use models to render phenomena. Our models are based on deep neural networks (DNN). We train models for multiple zoom levels while allowing users to interactively engage with the visualizations using panning, zoom-in, and zoom-out operations. Constructing models at different resolutions that are aligned with zoom levels allows us to reduce model complexity while ensuring fidelity.

Rather than exhaustively retrieve all data that must be visualized, we rely on retrieving a fraction of the dataset; this fractional dataset is then used by our models to render phenomena while ensuring fidelity and preserving interactivity.

Inferences are performed during the critical path of visualizations. During inferences, we seed the model(s) responsible for rendering tiles within a viewport with a fraction of the ground truth data. Our models work in tandem with a distributed, in-memory cache to support the effective seeding of models during rendering operations.

Training deep networks is resource intensive. We leverage transfer learning across the different zoom levels to manage computational complexity. In particular, the structural composition of the model at different zoom levels is the same. The models are parameterized differently for different zoom levels. Our novel transfer learning scheme performs transfer learning across different refinement depths. This has two benefits: (1) the base model, at the coarsest resolution across diverse variations that occur at larger spatial scales, while (2) the refined models at higher resolutions account for subtle variations at the regional levels.

Together, our transfer learning scheme allows the models to generalize (to diverse input distributions observed over larger spatial extents), be accurate because they are fine-tuned to account for regional variations, and be resource efficient (eliminate duplicate processing).

An in-memory, distributed cache is used for model inferences during runtime. The distributed cache performs prefetching and evictions based on spatiotemporal cubes. The dimensionality of the cube is based on the tile size for the highest resolution and the typical size of the temporal window used for drill-down and roll-up operations. Each cube is assigned a weighted score based on its past accesses and the likelihood of subsequent accesses. Crucially, because the weights assigned to individual scores are across accesses from diverse users, the eviction scheme accounts for usage across users.

Our benchmarks demonstrate the suitability of our approach to rendering visualizations; we discuss this in our benchmarks section. Consider exhaustively rendering phenomena by retrieving all observations; this takes 56.1805 secs to complete the fetch-and-render operations. Our approach of seeding machine learning models while incorporating caching and prefetching mechanisms renders the same phenomenon

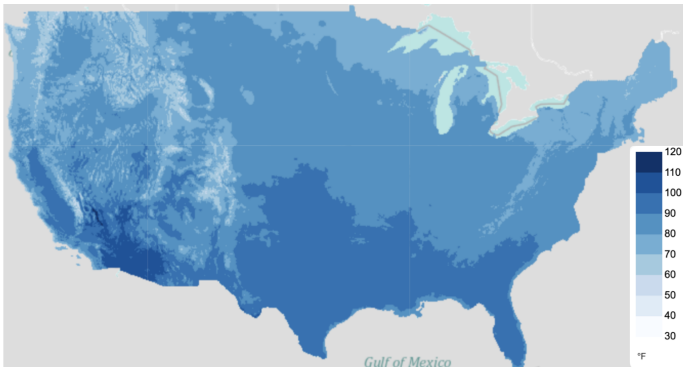


Fig. 1: Rending maximum air temperature phenomenon over CONUS exhaustively takes up to 56.1805 secs.

in 4.3043 secs, a 92.3% reduction in rendering times while preserving a high PSNR accuracy of 38.7 dB.

D. *Paper Contributions and Translational Impact*

Our methodology for rendering spatially evolving phenomena includes the following contributions:

1) A novel scheme to alleviate network and disk I/O costs by leveraging models to render visualizations at scale.
2) Our transfer learning scheme facilitates the effective reuse of network layers across different zoom levels. This allows our models to generalize better while preserving fidelity while utilizing resources efficiently.
3) A distributed, in-memory caching scheme that facilitates eviction and prefetching decisions across different user sessions and exploration trajectories.

Translational impact: The proposed methodology does not make any assumptions about the underlying spatial referencing system. As such, it is broadly applicable to gridded datasets that arise in other domains such as computational fluid dynamics. Similarly, gridded datasets occur in non-terrestrial settings such as atmospheric and oceanic phenomena; this work translates to those as well.

E. *Paper Organization*

The rest of the paper is structured as follows. Related research works in image super-resolution GANs, autoencoders, inpainting techniques, caching, and rendering schemes for visualization are discussed in Section II. The approach including model architecture, back-end caching, and transfer learning schemes are covered in Section III. Section IV covers the experimental setup, network, disk I/O reduction, and model performances deployed using caching schemes. In Section V, we conclude our work and present future directions.

II. Related Work

Front-end visualization - Visualizing voluminous geospatial datasets can incur high latency, network I/O, and memory consumption for transferring data across servers and client machines. Kraak et al. [1] provide techniques to optimize the visualization of geospatial data. In paper [2], authors leverage data reduction techniques to project high-dimensional data to 2D planes for fast visualizations. Similarly, in paper [3],

authors leverage the concept of self-organizing map methods for dimensionality reductions and clustering. Paraview [4] is a visualization tool for voluminous datasets that allows clients to run tasks in parallel over distributed memory.

Caching - Visualization can also be accelerated by deploying caching mechanism [5], [6] in the back-end servers or client's machine. In the paper [7], authors present an in-memory caching mechanism distributed across machines that performs hierarchical aggregation for serving fast queries. Authors in [8] propose intelligent caching strategies using replication and distributing data using spatial properties. In another work [9], a predictive model was presented to prioritize caching of geospatial tiles by learning high-priority geographic regions. Such caching schemes are often combined with streaming [10] and sketching [11] to facilitate rapid visualizations.

Autoencoders - Autoencoders are machine learning algorithms utilized for representational learning. Autoencoders are widely used for reducing data dimensionality and noise in inputs. There are many different variants of autoencoders developed for a variety of applications. Such as image compression [12]–[15] where the model learns to interpret the relation between input and output using an informational latent vector. Another variant is de-noising autoencoders where the model is generalized well enough to regenerate a noisy input vector and it is widely used in speech recognition [16], [17].

An autoencoder [18] comprises an encoder, a decoder DNN, and a feature vector. Encoders are utilized to extract the latent vectors (features) from a given set of input features. The extracted features typically capture nuances of the input image such as color hint, object position, etc. The bottleneck latent vectors are forced to learn a compressed form of the input and can vary in size depending on the complexity of the input dataset and desired compression [19], [20]. The decoder network reconstructs the latent vector back to its closest true representation. The major drawback of autoencoders arises from storing these latent vectors on the client machines for reconstruction while assuring high fidelity.

Super-resolution GANs - Super-resolution GANs are ML models, which we also deploy in our approach. Super-resolution is an image reconstruction technique to enhance the quality/resolution of a low-resolution input image. There are several areas in which super-resolution methods are applied such as medical imaging, video processing, satellite imageries, etc [21]–[24]. The super-resolution can be performed by utilizing multiple shifted low-resolution images capturing different regions in a low-resolution image or employing convolution layers in DNN models.

A high-performance boost was observed when these networks were incorporated with Generative Adversarial training [25]. GANs comprise two separate neural networks, a generator to predict high-resolution images and a discriminator to distinguish between the generated images and corresponding true values to push the learning of the generative model. Models [26] enhance the perpetual quality of the image by reducing artifacts and blurriness. The GAN models are usually memory intensive and require high computational requirements.

In [27] authors propose a memory-efficient method to deploy these GAN models in mobile devices using a multi-scale feature aggregation network while maintaining the desired accuracy. The major drawback of many of these networks is the dwindling perpetual quality of images when performing super-resolution of more than x4 resolution.

Inpainting ML models - Our approach can also commune to inpainting techniques [28], [29] which involves ML models to reconstruct missing patches in the input image. These techniques are proven to perform well for tasks such as removing clouds from satellite images, image restoration [30], or object removal. The patches can be filled by capturing patterns or textures from similar neighborhood regions. However, they eventually suffer from failure in capturing global patterns [31], [32]. Some works [33], [34] involve using GANs with dilated convolutions and ResNet architectures to improve image quality. In [35], authors suggest using partial convolutions for reconstructing irregular patches. While in [36] authors improvise the UNet architecture to reconstruct patches in the corrupted images. Our work involves filling the missing regions at regular space intervals and thus is more acceptable as a super-resolution application.

III. Methodology

Our proposed methodology encompasses models, data structures, algorithms, and caches working in concert with each other to ensure effective visualizations at scale. In particular, these include:

- Data wrangling and sculpting operations including construction of GeoTIFFs from gridded datasets.
- Designing deep learning models to render spatiotemporally evolving phenomena.
- Designing transfer learning schemes that manage computational complexity and accelerate the training of models.
- A distributed cache to facilitate fast, effective seeding of models during inferences.
- Dynamic visualizations at the client side.

The distributed cache is a memory resident. It incorporates mechanisms to perform evictions and prefetching based on the access trajectories.

A. *Data Wrangling: [RQ-1]*

Our methodology targets the visualization of gridded datasets; each grid point is identified by $\langle lat/long \rangle$ coordinates and includes a vector of observations alongside a timestamp. In this study, we consider one of the most well-known gridded datasets, MACA Multivariate Adaptive Constructed Analogs) [37]. The MACA dataset represents an amalgamation of over 20 global climate models (GCMs) downscaled to 4km (1/24th degree) spatial resolution and representing different outcomes for the Radiative Concentration pathways for greenhouse gas emissions. We consider the RCP 8.5 trajectory. We consider data encompassing projections for the years 2023-2030 for the continental United States (CONUS).

Figure 2 shows the meteorological parameters of the dataset that we focused on and the range of their values. This includes

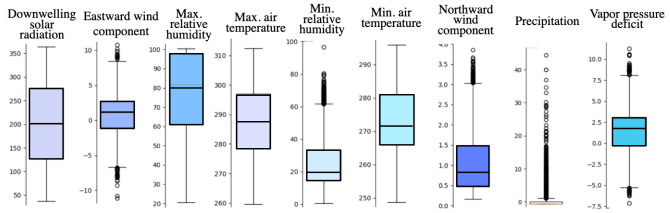


Fig. 2: Range of values for various meteorological parameters at sampled locations from the MACA dataset.

multiple parameters such as maximum and minimum temperature, maximum and minimum relative humidity, precipitation accumulation, downward surface shortwave radiation, wind velocity, and specific humidity.

Tiling scheme. An integral part of our methodology involves constructing GeoTiff images out of gridded datasets and partitioning them into smaller spatial extents of 256 x 256 km. The raw gridded MACA dataset for CONUS is stored in an Apache Druid datastore. Druid includes support for query plan optimizations over large data collections. We construct 3D images/GeoTIFFs using latitude and longitude information provided for each grid point at 4 km resolutions alongside 10 meteorological variables.

Zoom levels. Within the viewport, the canvas is partitioned into tiles. The size of these tiles corresponds to the spatial extent under consideration. Visual explorations result in changes to the tiles being rendered, the spatial extents that the tiles correspond to, zoom levels, and the resolutions associated with the visual artifacts.

During visual explorations, we support (1) spatial navigations encompassing zooming-in and zooming-out, (2) temporal navigations encompassing drill-downs and roll-ups over time for the entire viewport, and (3) panning operations where a user might navigate sideways in space and/or time. Consequently, tiles exist across space and time; these tiles are materialized (i.e., computed and rendered) based on the user's exploration trajectories.

From the perspective of how tiles occur in space-and-time, there is a great deal of structural similarity. Zooming in renders spatial extents at increasingly higher resolutions. The spatial extent being rendered is smaller, but the rendering themselves gets more detailed e.g., topographical or structural characteristics; similarly, during zoom-outs, while one set of tiles is fused into a large tile, multiple new tiles at coarser resolutions are rendered to complete the viewport. Hierarchically, the zoom-in/zoom-out (and the temporal drill-down/roll-up) result in tiles being created, fused, and/or new ones brought in. This is analogous to fractals where self-similar patterns appear at different scales.

To ensure residency of the DNN models within the GPU RAM during training, we partition the GeoTIFF images into smaller tiles each of 256x256 pixels at 4 km resolution. As shown in Figure 3 (a), we start at resolution level 1 (coarsest spatial resolution), where the raw tile (256x256 pixels) is spatially sampled such that each pixel is 16 km apart, resulting

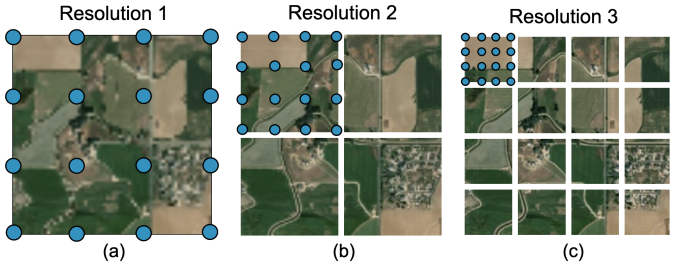


Fig. 3: Constructing images out of gridded dataset and breaking them into multiple tiles at different spatial resolutions. At resolutions 1, 2, and 3, pixels are spatially 16 km, 8 km, and 4 km apart respectively. Here, blue dots represent the fraction of data ingested by the model for inferring the complete tile.

in a tile of size 64x64 pixels. While refining tiles at resolution level 2, raw tiles are sampled to have pixels at an 8 km spatial distance as shown in Figure 3 (b). Merging four such nearby tiles each of 64x64 pixels in size at resolution level 2 results in the same image as in Figure 3 (a) but at finer spatial resolution. The last resolution level, level 3 is where the finest resolution of a given region is available; here, every pixel is 4 km apart as in the actual dataset. Merging 16 such tiles each of 64x64 size results in the same image as in Figure 3 (a and b), however at the finest spatial resolution possible.

Geocoding. At each spatial zoom level, the gridded dataset tiles are partitioned into smaller tiles with 64x64 pixels. The model is trained to reconstruct high-resolution meteorological values for a fixed-size spatial extent. The size of the spatial extent is based on the memory and computational footprints associated with model training. Our partitioning scheme incorporates a deterministic scheme where each such tile is associated with a character or string of characters called a *geostring*. This allows us to partition spatial extents into a set of equal-sized non-overlapping bounding boxes. Proximity (or closeness) in the *geostring* space translates to spatial proximity at any zoom level. As we refine the spatial resolution of the tile, we subsequently divide the *geostring* into 4 smaller bounding boxes and attach a single character - 'a', 'b', 'c', or, 'd'. The partitioning scheme is hierarchical and maintains parent-child relationships, for say, part of the observations captured by geostring 'f' at resolution level 1, are represented at finer resolution by geostring 'fd' at resolution level 2 and geostrings 'fda', 'fdb', 'fdc', and 'fdd' at resolution level 3 as shown in Figure 4.

B. Designing Deep Neural Network (DNN) to render phenomena: [RQ-1]

Our methodology involves seeding the models with a fraction of the actual data. This allows us to alleviate expensive disk and network I/O requirements by leveraging DNN models to render the phenomena. Rather than using the entire set of available observations, we train DNN models to super-resolve and learn non-linear interactions from a small fraction of available data to render the finest resolution tile of size 64x64 pixels for 8 meteorological parameters.

DNNs. In this section, we discuss modeling the network

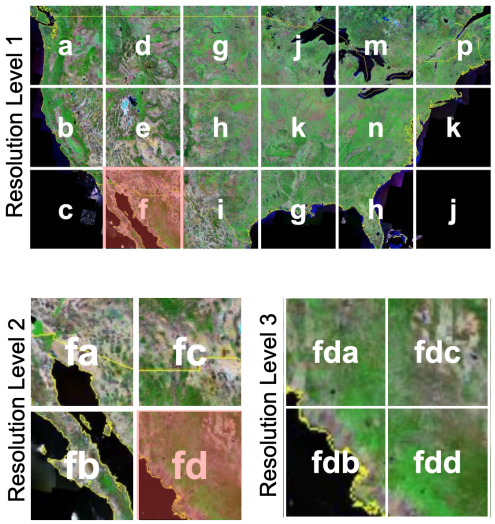


Fig. 4: Hierarchical partitioning of CONUS into non-overlapping equal-sized tiles at three different zoom levels.

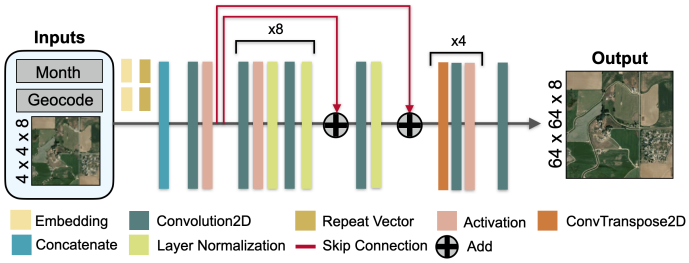


Fig. 5: The architecture for our super-resolution DNN. Repeated cells of convolutional layers with skip connections followed by downscaling blocks to increase spatial resolution.

which ingests a fraction of data to infer real-time meteorological information by leveraging self-supervised learning. The core of our deep neural network architecture is inspired by SRGAN [38], which is a super-resolution generative adversarial network as depicted in Figure 5. We provide the model with a location hint by generating an embedding vector for the geostring associated with the input tile. The model also ingests the temporal hint provided as an embedding of the associated month. This helps in learning interactions between the labeled pairs - input and full output images, based on a particular region and seasonality. One such example is the low temperature in the winter months or the high humidity during the rainy seasons. This allows the model to generalize better on unseen data (e.g., from different years than training data but during the same season). Both the spatial and temporal hints are passed through the repeat vector layer and reshaped into 4x4x1 dimensions so that they can be merged with low-resolution input data (4x4x10).

The low-resolution input image is generated by selecting every 16th pixel of the tile in both the x and y dimensions for each meteorological parameter. Training the DNN model with limited true information results in very low network I/O performed to get these values from our back-end distributed cache to the client's device. The model learns to map as low as 0.003906 fractions of true values (16 points) to generate high-resolution visualization of the tile (4096 points). We also allow the single DNN model to train across multiple features of interest by leveraging the non-linear interactions occurring between these variables. The meteorological variables are often inter-connected; for example, the temperature is inversely proportional to relative humidity throughout the day. Providing the model with all parameters allows the model to learn inter-feature relationships. This low-resolution input image is concatenated with temporal and spatial hints and passed through pre-residual blocks comprising 2D convolution layers and a ReLU activation function. The input meteorological parameters are normalized and scaled individually between 0-1 for land regions. The ReLU activation function ensures that values emerging from the convolution layer do not saturate.

The pre-residual blocks are followed by 8 blocks of residual blocks connected through skip-connections to avoid vanishing gradients. The residual block comprises two convolution layers, layer normalization, and a ReLU activation function. The number of features in convolution layers is increased and kernel size is kept at 5x5 to extract low-level feature maps across the neighboring spatial region in the image in order to retain crucial information for higher spatial resolutions. Here, we perform layer normalization that normalizes the activations along the parameter/feature dimension instead of normalizing the batches. This is to account for the fact that each of the parameter values is at different ranges. Next, we perform upsampling of the image by consecutively increasing the spatial dimension of the features maps using a block of convolutional transpose layers and an activation layer to infer the full image with all 8 output parameters.

The number of output features at each convolutional, convolution layer, learning rates, and kernel size is fixed by performing hyper-parameter tuning using Hyperband [39]. Determining a robust set of hyperparameters is crucial for expedited model training and better performance. We leverage the Tensorflow Hyperband tuner which speeds up extensive parameter grid searches through adaptive resource allocation and early stopping to identify best-performing combinations.

Loss Function. Our DNN model's weight and biases get reparameterized using an adaptive loss function. The MACA dataset we consider is available for terrestrial regions over CONUS. To deal with missing observations, we force our model training to learn from a land mask. The mask pixel is set to 0 if there is no observation available for that location, otherwise, to 1. Therefore given a model-predicted image and target image, we allow the model to pass the errors through back-propagation based on only the MSE errors (Mean-squared errors) over land pixels only and update the gradients accordingly. This is shown in the equation below -

$$Loss = \frac{\sum_{n=1}^{64*64}(|Predicted_n - Target_n|^2 * land\_mask)}{64*64} \quad (1)$$

Distributed Training. To orchestrate model training and targeted refinements across different spatial scales, we leverage

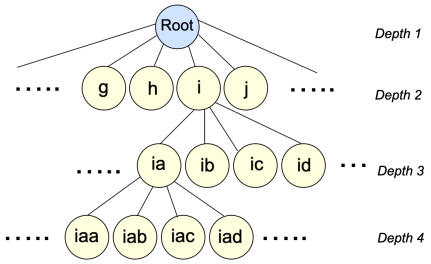


Fig. 6: The tree structure fixed at depth of 4 maintained to keep track of model instances prediction accuracies and a pointer to the model image in memory.

a tree-based data structure, the refinement tree. The refinement tree encapsulates information about the spatial extents (identified using geostrings) for which model instances were created and the hierarchical relationships, via transfer learning, that exist between the parent and child. Each node includes metadata information regarding the model's performance and informs refinement efforts. The root node represents the entire spatial extent under consideration and serves as the conduit for all traversal operations within the tree. Traversing the root refinement tree at different depths represents models for progressively higher resolutions and smaller spatial extents. This is depicted in Figure 6.

Initially, the models associated with all nodes at depth 1 are marked with a testing accuracy of 0 dB PSNR indicating that the corresponding model instance needs to be refined. Model instances associated with a particular depth are trained in parallel across separate GPU cores over multiple machines.

Model instances that meet the desired accuracy thresholds are deemed performant and are not subject to further refinements. The desired threshold values are configured to conform with acceptable PSNR values for images within the range of 30 to 50 dB (the highest possible value) [40]. In our methodology, PSNRs were set at 30 dB, 34 dB, and 36 dB for resolution levels 1, 2, and 3 respectively.

The model structure and weights are then used to transfer learning and warm-start model training for the smaller spatial extents. During transfer learning the latter upscaling stages are deemed trainable and the process repeats till we spatially cover all refinement levels. Model instances for the smaller spatial extents are fine-tuned with data that are specific to the extent under consideration. Once all nodes are deemed to have met the accuracy thresholds, the training phase is completed. The models are then used for inferring.

C. *Hierarchical Transfer Learning: [RQ-2]*

We design DNN models for different resolutions that are aligned with the diverse zoom levels that we support. This reduces model complexity while also ensuring high fidelity. Models are trained separately for each refinement/zoom level. At any zoom level, we identify spatiotemporally proximate data during model training and inferences to seed the model. At any zoom level, there are multiple instances of DNN models trained to achieve the highest fidelity for every location.

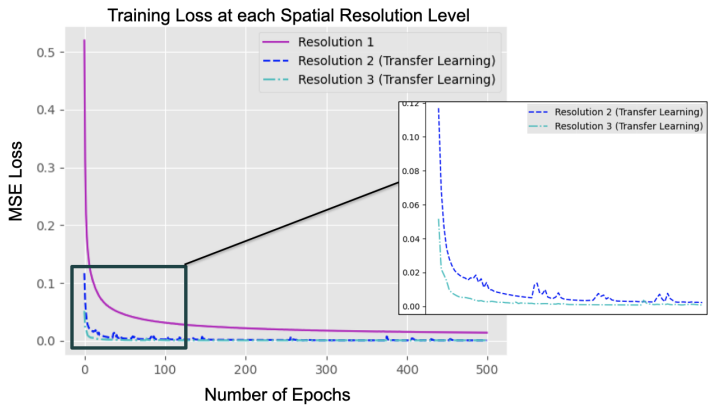


Fig. 7: MSE Errors while training the DNN models using our spatial hierarchical transfer learning approach for each refinement level.

We start our model training at zoom level 1 which renders the phenomenon at the coarsest spatial resolution, here we separate CONUS into three distinct regions requiring the seeding of models. Each of these three models is trained in parallel over multiple machines. The weights and biases are initially parameterized by random Gaussian distribution vectors. Once all the models reach the threshold of testing PSNR (Peak-to-signal ratio) accuracy of 30 dB, we halt the training.

The trained models from the previous zoom level l are then used to initialize the weights and biases at the subsequent zoom level. At the next zoom level (l+1) and similarly (l+2), we render full tiles at a much finer spatial resolution, by reducing the spatial extent of each bounding box while increasing the spatial resolution of the inferred sample. We group spatiotemporally proximate regions into a much higher number of clusters, this is to ensure that seeded models fine-tune better for diverse geo-locations and generate outputs at higher fidelity as the same 64x64x8 image now carries much finer meteorological information.

The improvement of transfer learned models by passing model parameters hierarchically across spatial resolutions is evident from Figure 7, where we show the reduction in MSE loss over training progression for models seeded for zoom level 1 (trained from scratch) vs loss errors for subsequent zoom levels after warm-starting the learning from previous zoom refinement level. While the model for coarsest resolution takes much longer to converge and has high errors (0.5 MSE errors) at the start, models at subsequent zoom levels 2 and 3 benefit from transfer learning and the non-linear patterns observed at those spatial extents at coarser levels. The learning begins at very low MSE errors (0.1 and 0.05 MSE errors) and converges much faster at every consecutive spatial zoom level.

Our overall spatial transfer learning scheme is hierarchical with each refinement being performed on a finer, more-proximate zoom level. The transfer freezes the weights and biases of the initial residual blocks of the model, and only the final layers of the DNN are trainable. This results in expedited training of the model while improving the quality

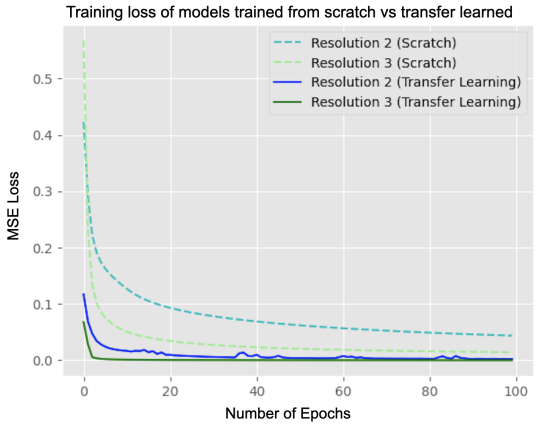


Fig. 8: A comparison of MSE losses during our DNN model training with and without spatial transfer learning at zoom levels 2 and 3.

of inferences. Before training the individual models at zoom levels l+1 and l+2, we identify regions where our parent model produces images that exceed the image quality thresholds and are thus deemed performant for generating inferences for that spatial extent. The scheme further allows us to reduce the number of regions (and the new models) that need to be fine-tuned thus facilitating a reduction in training times. In Figure 8 we provide training loss from models at zoom levels l+1 and l+2 trained by "cold-start" weight initialization vs getting "warm-starts" from coarser spatial resolution models. Models trained from scratch have much higher error loss and take an extensive number of iterations to converge when compared to the transferred learned model. Further, models at l+2 spatial zoom level entail reduced training times with higher accuracy when contrasted with models at zoom level l+1. In Section IV, we provide empirical benchmarks and profile resource requirements for training models at each zoom level.

D. Back-end Distributed Caching: [RQ-3]

We designed a distributed cache service that enables fast querying over the gridded datasets. The cache is distributed, memory-resident, and backed with eviction and prefetching schemes that ensure that only a small subset of the data is preferentially selected for inclusion in the case. The distributed cache presents a service interface to the visualization clients and comprises three components: (1) A REST API that intercepts queries issued by the client to the cache/database service, 2) A coordinator node that reads and writes data to the storage nodes, and 3) a query interface to issues queries to the primary, on-disk database during cache-misses and prefetching, (see Figure 9). We leverage the Python Flask library [41] and the Redis in-memory key-value store [42] to implement the REST API and distributed cache respectively. Our primary, on-disk data store comprises a distributed cluster of Apache Druid [43] nodes.

Data items within the cache are organized as data cubes so called because they encapsulate data from a 2D spatial extent alongside time. Each cube represents data from the

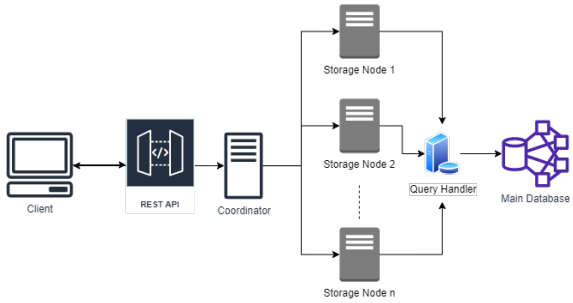


Fig. 9: Components of the back-end distributed caching.

finest resolution geostring and the temporal scope of the cube is based on the average drill-down/roll-up along the temporal axis as discussed in III-E. The size of the cubes allows us to address boundary conditions during incremental panning operations. Additionally, the caching service supports batched aggregate operations that process read/write operations for multiple days i.e., multiple cubes with a single call to the storage server, thus reducing the number of network I/O operations. Data evictions are performed in units of data cubes as well.

The interactions of the aforementioned components take place in multiple stages. When the client issues a query containing the geostrings and the corresponding timestamps, the REST API intercepts the requests and issues a call to the caching service. The caching service then looks up the data record to check if it is present in the cache. If the record is found (a cache hit), the service returns the record back to the client. Otherwise, the caching service issues a query to the main database and sends the data queried from the database back to the client machine, while also caching the newly queried record for future use. The cache uses an LRU-based eviction scheme to make space available for newly fetched items.

Data Prefetching. In addition to servicing queries as they occur (the reactive mode), the caching service also incorporates a proactive mechanism that tries to anticipate queries that may be issued in the future and prefetch any necessary data by storing them in the cache. Anticipatory prefetching ensures that any I/O that may need to be performed during cache misses is performed away from the critical path during interactive visualizations.

Our prefetching schemes target both the temporal and spatial dimensions of the dataset.

- **Spatial prefetching**: The client usually issues a query for a viewport containing multiple spatial extents (geostrings). When a query comes in, we prefetch and store data specific to proximate spatial extents. Such prefetching allows effective support for typical spatial panning and zooming operations.
- **Temporal Prefetching**: We supplement our spatial prefetching with prefetches in temporal proximity with 2n days' worth of data (+/- n days around the original query timestamp). The variable n is a configurable parameter. We also include support for prefetching data in temporal increments of weeks or months around the specified time point.

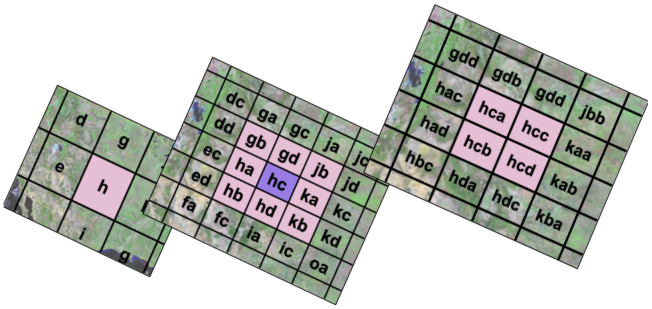


Fig. 10: Spatial prefetching performed by the back-end cache. The region highlighted in purple represents one of the geostring included in the client's initial query over a random viewport. The pink regions depict the spatial extents that are prefetched at the same resolution level along with regions at finer and coarser spatial resolutions.

Scalability. The caching service is distributed over a cluster of 12 storage nodes. To enable horizontal scaling, we implemented functionality to dynamically add or remove storage nodes based on loads at the backend services. The service is also capable of rebalancing (re-sharding) based on data distribution and dispersion in the cache. Rebalancing operations are invoked periodically at configurable intervals, based on sustained utilization spikes at certain nodes.

Eviction schemes. The caching service leverages the basic LRU scheme and updates metadata to ensure targeted evictions of records. We allow users to configure the caching service with evictions based on LRU with optional support for an LRU scheme that incorporates elements of LFU (Least frequently used to achieve finer control over the eviction process. We maintain a stateful variable to record the number of times a record has been accessed. At the time of eviction, we check each element's access frequency against a preset threshold which allows us to keep the most frequently accessed records in the cache, thereby overriding the primary eviction policy for a select number of records.

E. Interactive Visualization: [RQ-2]

Our overall system comprises the front-end client machine that issues queries over a viewport. The viewport is partitioned into fixed-size tiles for each spatial resolution. At spatial resolution 1, the viewport tiles are represented by a 416 km x 556.6 km area, at resolution level 2 it is 227 km x 417 km and at resolution level 3 viewport captures an area of 138 km x 283 km. Given the lat/long bounds of the viewport, our framework identifies the geostrings that are enclosed by the bounding box.

Users have the ability to request time-series data for any timeframe. This information is encapsulated in a JSON object and passed as a request to our back-end cache. The query response is the fraction of true values for each geostring for n number of days. This response is then ingested by our DNN models deployed on the client's machine. The inferences of the model are stitched together to render the phenomena over the viewport.

Model Pruning. To render visualization at the client's machine, it is crucial that the model inferences are performed fast while assuring the low memory footprints and high quality of inferences. The client machine can be a commodity machine with a single GPU core, fitting the model instances in machine memory is crucial for our framework. To ensure space efficiency and reduce the memory footprints of our model, we leverage Tensorflow optimizations post-model training [44]. In particular, these optimizations allow us to prune the weights gradually for all layers in the model which don't affect the model's outputs. The models are fine-tuned for 10 epochs at 5% increments in the amount of pruning (starting with pruning 40% weights) until it reaches 80% pruning. At each step, the weights of the newly pruned model are adjusted to attain desired inference accuracy. Finally, we use the Tensorflow Lite library [45], [46] which converts the model weights into Tensorflow Lite graphs that are lightweight models with weights at 8-bit precision. The optimization aids in reducing latency and inference time, especially for mobile applications. In Table II, we report the amount of compression achieved for all models at different spatial resolutions. We achieve x23, x19, and x16 fold reduction in memory footprints at resolution levels 1, 2, and 3 respectively.

User query design. To assess the performance of our framework, we synthesize queries that are aligned with visual analytic operations over large spatial-temporal extents [47]. These queries include multiple types of operations such as panning, zooming (along the spatial dimension), and drilling-down/roll-ups along the temporal dimension. Spatial panning includes moving the viewport around the target region at the same spatial resolution level to explore a neighborhood. We also zoom in/out spatially over a region to render finer or coarser-grained images respectively. For temporal panning operations, users can query multiple days' worth of information. Our system is capable of rendering visualization for 7, 14, and 21 days. The temporal operations incorporate configurable support for weekly (+/- 3), monthly (+/- 5), and yearly (+/- 3) temporal jumps into the past or the future.

IV. Systems Benchmarks

A. Experimental Setup

We profile our methodology over a cluster of machines. The backend distributed storage that hosts our MACA dataset is dispersed over 50 machines. Our distributed, memory-resident cache encompasses 12 machines. We synthesize client queries using a service that is deployed on each client machine and initiate a set of concurrent queries that are aligned with common user interaction patterns. Each of our client services runs on machines with Xeon E5-2620, 64 GB Memory coupled with a single Quadro P2200 GPU (5GB of memory with 1280 cores).

Throughout the experimentations for rendering phenomena, we report performance across three spatial resolutions as discussed in Section III i.e., resolution level 1 (coarsest with 16 km pixel spacing), resolution level 2 (medium with 8 km pixel spacing), and resolution level 3 (finest with 4 km pixel spacing). To profile our system, we design queries

Resolution Level	Total Training Time (in epochs)		Number of training samples required	
	Scratch	Transfer Learned	Scratch	Transfer Learned
Level 1- Coarse	7000	-	23,725	-
Level 2 - Medium	8000	700	76,650	25,570
Level 3 - Fine	10,000	500	262,800	127,750

TABLE I: Comparing the training requirements for models at each spatial resolution level in terms of epochs and the number of training samples models got trained.

Resolution	Time taken to render gridded dataset for n days			No. of model instances	Total memory consumption	Total memory consumption
	1 Day	7 days	14 days			
Level 1- Coarse	31.2 ms	88.6 ms	142.6 ms	3	1.575 Gb	68.47 Mb
Level 2 - Medium	32.1 ms	106.6 ms	193.16 ms	6	3.15 Gb	165.78 Mb
Level 3 - Fine	36 ms	189.1 ms	364.1 ms	10	5.25 Gb	328.125 Mb

TABLE II: Comparing the inference time for rendering tiles over CONUS for each spatial resolution over 7, 14, and 21 days. The table also contrasts the number of models deployed at each resolution level and their memory consumption.

that reflect typical user query patterns such as spatial zoom-in/outs, panning around a region, and exploring temporal drill-downs/roll-ups (jumps w.r.t weeks, months, and years). Lastly, to gauge the efficiency of our back-end cache, we contrast performance in three different settings - (1) bounding box queries that exhaustively fetch all the gridded points over the queried lat/long bounds at 4 km spatial resolution, (2) Without caching where back-end responds with a fraction of data points essential for performing model inferences but without caching and prefetching operations, and (3) With caching and prefetching only a fraction of data.

B. Modelling and Transfer learning scheme: [RQ-1, RQ-2]

One of our objectives is to profile model training requirements to render high-quality phenomena with and without the transfer learning scheme discussed in Section III-C. The training cost of the model is directly associated with the memory footprint of the model, the number of training samples ingested which can further be data parallelized using distributed training across machines, and the number of iterations required to reach the desired accuracy threshold.

As seen from Table I, the number of training iterations reduces 11-fold and 20-fold for resolution levels 2 and 3 respectively when utilizing our spatial transfer learning scheme. This directly affects the training time requirements for transfer-learned models, as models are pre-initialized with weights over diverse and coarser resolutions spatial extents but still incorporate phenomena hints for the localized region. The reduction in training time is also contributed by the fact that while performing transfer learning we first estimate the accuracy of the parent model over finer spatial extents. At some of the spatial extents, the parent models are deemed to be the performant, and training data for those extents need not be accessed. At resolution levels 2 and 3, the number of training samples is reduced by one-third and one-half when compared to training models from scratch. At resolution level 1, each tile covers larger spatial regions, the number of samples is very low and we stop training models at a relatively low accuracy threshold to render coarser images.

In Table II, we report inference times for models to render phenomena at each resolution level for a day, a week, and two weeks. With increasing resolutions, CONUS is divided into a higher number of smaller spatial extents which entails a greater number of models. We also report the number of models deployed at each resolution level. At level 1, three models suffice to reach threshold accuracy across all the spatial regions. At finer resolution levels, 6 model instances are needed at level 2 and 10 for level 3. For any resolution, models

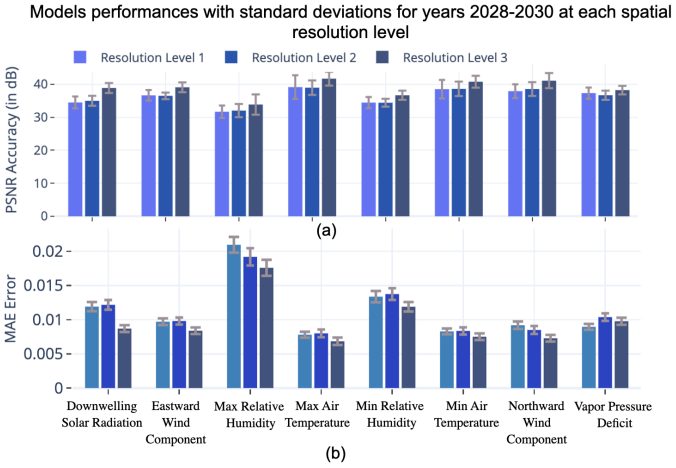


Fig. 11: (a) PSNR accuracy and (b) MAE errors on testing samples with error bars for each of the 8 meteorological parameters.

that are being used are compressed and memory-resident.

We also compare the quality of inferences made by models across a variety of meteorological parameters (shown in Figure 11). We train models to learn inter-feature dependencies over multiple phenomena. A key aspect of our methodology is to render phenomena with high fidelity. From the figure, we observe that models perform equally well on all the parameters with an average PSNR accuracy of above 30 dB. We observe slightly high errors for the maximum relative humidity parameter due to a higher variance in the original raw values.

In Figures 12-15, we compare the perceptual quality of the model inferences to render the visualization of four the meteorological phenomenon alongside the actual sample points at different resolution levels. At each resolution level the model inferences are stitched together to render tiles over CONUS or a viewport on the client's machine. At resolution level 1, the visualization has slightly low perpetual quality with high variations along the borders of the smaller spatial extents or tiles. As the resolution of inferences is refined, the visual quality of the phenomenon is also improved and becomes indistinguishable from real samples at the finest resolution.

The observation can also be reinforced by Table III, where we report the testing PSNR accuracy and mean absolute errors for tiles captured from the timeframe that is unseen by models during training. The PSNR accuracy achieved is 36.2 dB, 36.34 dB, and the highest with 38.78 dB.

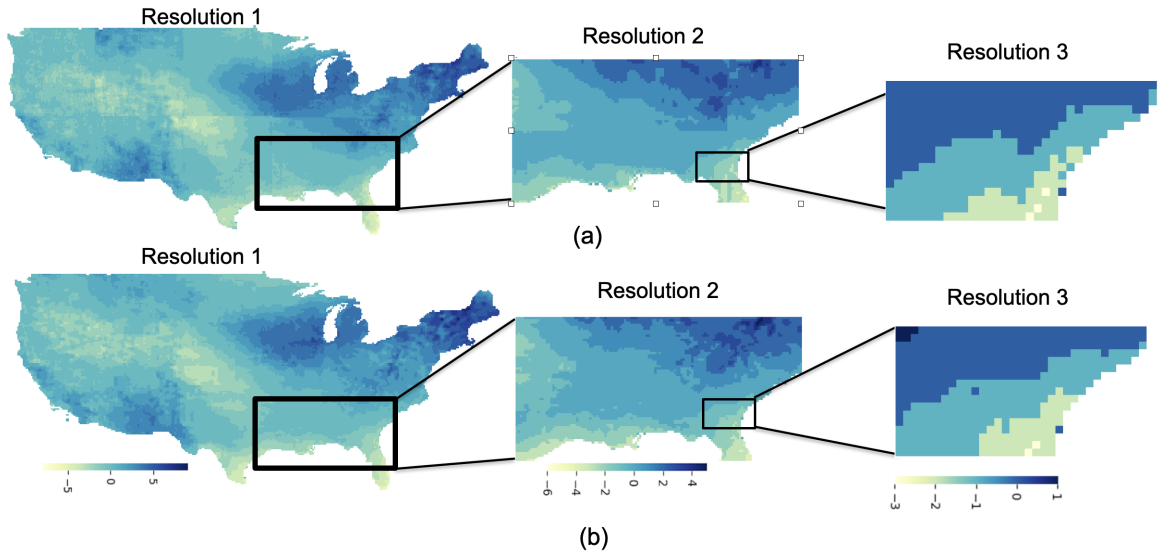


Fig. 12: Visualizing the vapor pressure deficit phenomenon over CONUS at resolution level 1, and smaller viewport at subsequent resolution levels. It measures the difference between the amount of moisture in the air and the amount of moisture the air can hold when it is saturated at 2m above the surface of the earth. (a) Inferences predicted by models. (b) The actual sampled image without model deployment.

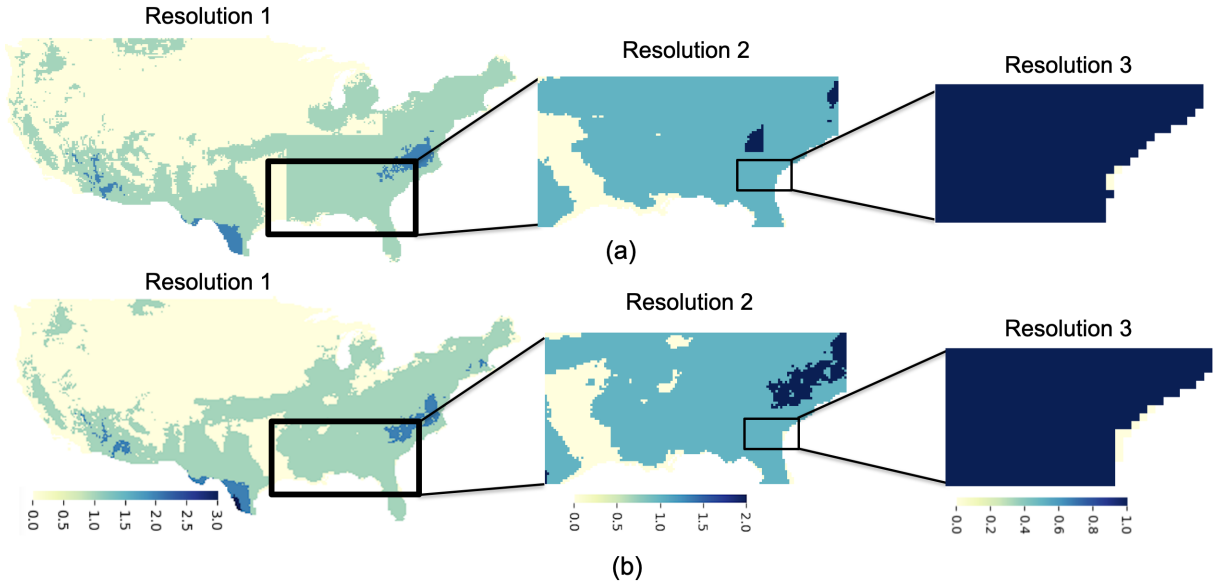


Fig. 13: Visualizing the northward wind component at three resolution levels. It depicts the wind speed in meters per second in the northward direction at 10m height above the surface. (a) Inferences predicted by models. (b) The actual sampled image.

Resolution Level	PSNR (dB)		MAE	
	Accuracy	Standard Deviation	Error	Standard Deviation
Level 1- Course	36.26	2.15	0.011275	0.000622
Level 2 - Medium	36.34	1.7	0.011270	0.000696
Level 3 - Fine	38.78	1.87	0.009750	0.000611

TABLE III: The testing accuracy of models at each resolution level for years 2028-2030 in terms of PSNR and Mean Absolute Errors along with standard deviations.

C. *Interactive queries: [RQ-3]*

To measure the overall performance of our proposed work, we report the turnaround time to service an end-to-end client query over a viewport for 7 days. This also includes the time taken to infer and render the phenomenon. We also contrast the quantity of data requested from the back-end server for both the bounding box queries and our approach.

In Table IV, we observe that there is a 99.96%, 99.86%, and 99.55% decrease in the number of data points retrieved by the back-end service for each spatial resolution exhaustively versus our approach. Similarly, we measure the turnaround time with and without caching and prefetching and compare it with a bounding box query. There is a 96.88%, 90.46%, and

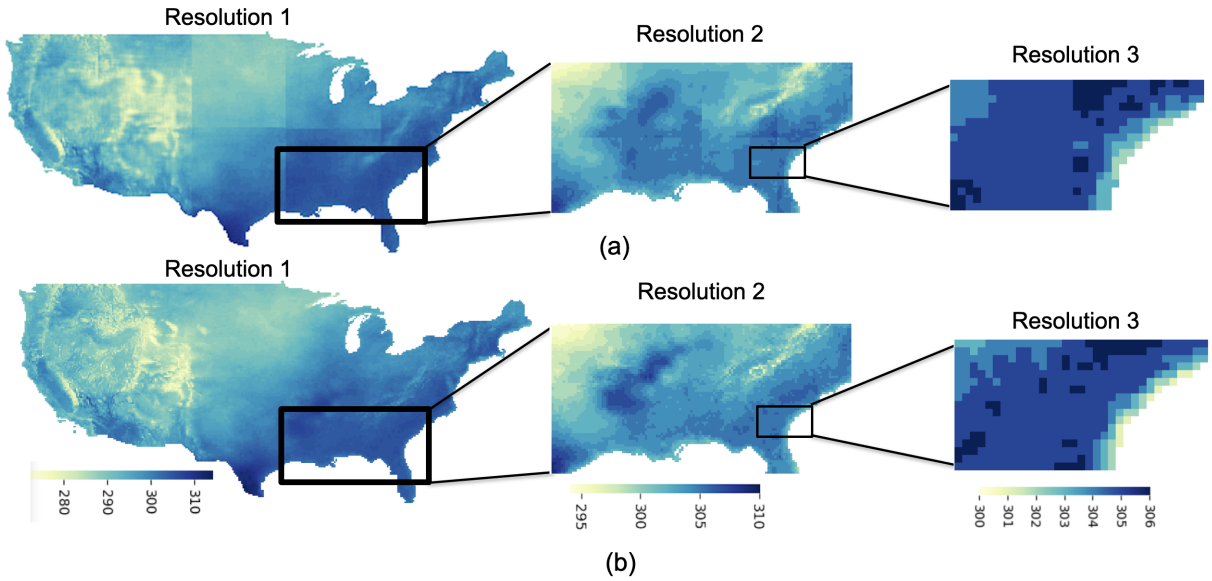


Fig. 14: Visualizing the maximum air temperature over CONUS. (a) Inferences predicted by models. (b) The actual sampled image without model deployment.

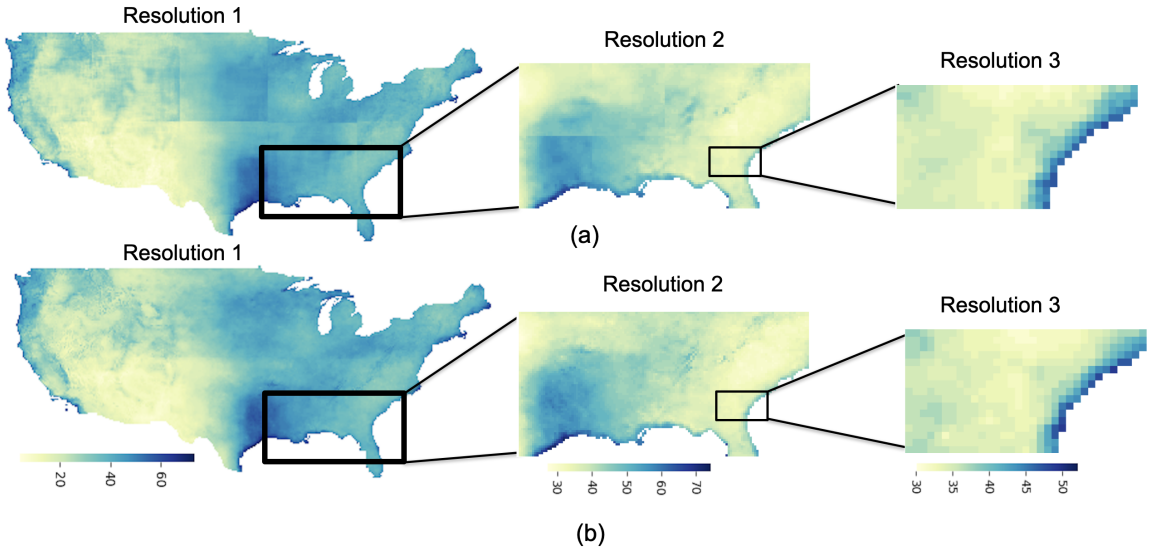


Fig. 15: Visualizing the minimum relative humidity. It is the lowest value of relative humidity in a day, which is measured as the amount of moisture in the air relative to the maximum amount of moisture the air can hold at a given temperature. (a) Inferences predicted by models. (b) The actual sampled image without model deployment.

79.32% reduction in time taken for rendering visualization at each resolution level without cache but with only retrieving a fraction of data from Druid. At finer resolutions, the number of queried data points also increases, as the viewport covers more tiles but at smaller spatial extents compared to exhaustively requesting all the points. With caching, partial data is retrieved from the memory of the back-end server nodes, while in the case of cache-miss, the remaining data points are accessed through the Druid data store. With the prefetching and caching scheme in place, we observed 65.26%, 78.089%, and 65.023% time reduction compared to without cache.

Resolution Level	Bounding Box		Our Method		
	No. of points queried	Time Taken (secs/query)	No. of points queried	No cache - Time Taken	With cache - Time Taken
Level 1- Course	999,775 points	70.78108	336 points	2.2080 secs/query	0.767 secs/query
Level 2 - Medium	568,617 points	46.296098	784 points	4.4133 secs/query	0.967 secs/query
Level 3 - Fine	405,104 points	40.10531	1792 points	8.2912 secs/query	2.9 secs/query

TABLE IV: Given a viewport comparing the number of points queried at each spatial resolution for 7 days and the time taken to respond to the client's query, perform model inferences, and render the phenomenon in diverse back-end settings.

Figure 17 shows the average completion time for the three main types of queries we utilize on five different variations of user queries (drill-in/out and temporal jumps). Overall, compared to the bounding box queries where all data points within

a given spatial area are retrieved from the main database, we observed a 65.02% reduction in completion time for queries with no cache access and a 99.44% reduction for queries with cache access and prefetching.

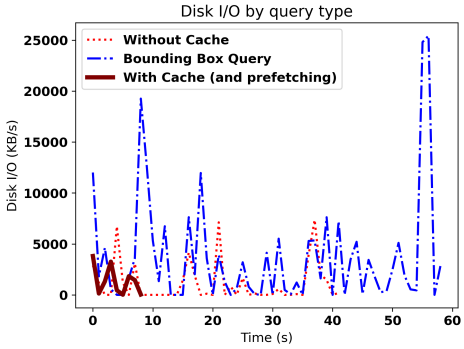


Fig. 16: Disk I/O - with and without caching

D. *Back-end Service: [RQ-3]*

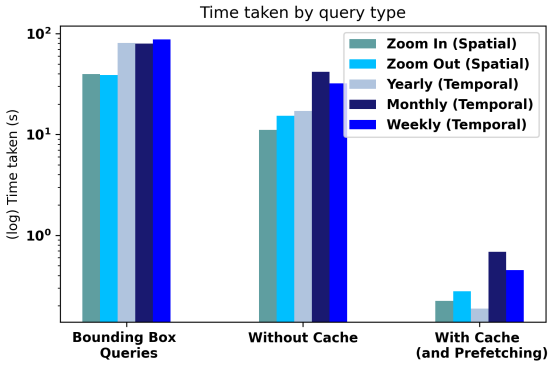
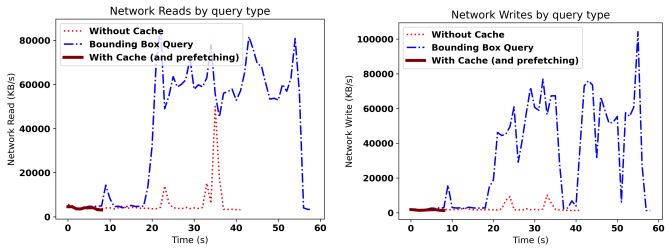


Fig. 17: Time taken by different query types (in log scale)



(a) Network reads (b) Network writes

Fig. 18: Network I/O - with and without caching

Percentage reduction	Disk I/O (%)	Network Reads (%)	Network Writes (%)
Without cache	78.24	89.45	94.76
With cache (and prefetching)	95.15	98.66	99.33

TABLE V: Reduction of I/O for query evaluations compared to exhaustive bounding box queries

We also profiled access on the back-end cluster during query evaluations. Figures 16 and 18 show the variation in disk I/O and network I/O operations respectively. The average reduction in the number of disk and network read/write operations for cached queries (with prefetching) and queries without cached access, compared to bounding box queries, are summarized in Table V. In almost all the cases, we were able to eliminate over 90% of the I/O operations required for querying by utilizing our distributed cache.

V. Conclusions and Future Work

In this study, we described our methodology to facilitate interactive visualizations over voluminous, gridded spatiotemporal datasets.

RQ-1: Having a multiplicity of model instances, each calibrated to particular spatial extents, allows us to reconcile heterogeneity in the spatiotemporal dynamics of phenomena. Considering a multiplicity of variables in the multivariate phenomena allows us to capture non-linear interactions that exist between features. Our methodology allows us to render phenomena with high fidelity of 38.7 dB PSNR.

RQ-2: Rather than train models from the ground up using cold-starts, we leverage our hierarchical spatiotemporal transfer learning scheme that allows us to target our model refinement efforts. Our methodology allows smaller spatial extents at higher resolutions to utilize significant portions of the parent network. Crucially, because the parent model was trained on data from a larger spatial extent we expect the refined models to generalize better. As our benchmarks demonstrate, our transfer learning scheme allows the models to train faster (number of epochs) and have better performance (36-39 dB).

RQ-3: Seeding models, during inferences, with a limited number of ground truth data allows us to reconcile the space efficiency of models with voluminous observational data. Having a distributed cache that preferentially prefetches and evicts data based on the user's navigational patterns allows significant reductions in the I/O (percentage reduction in I/O from benchmarks); especially, those that occur in the critical path during interactive explorations. Finally, leveraging model-specific compression optimizations allows us to preserve the space efficiency of models.

As part of future work, we will explore extensions to support non-gridded datasets such as fine-scale topographical information and shape files. In particular, we plan to explore diffusion methods in the design of our deep networks.

Acknowledgment

This research was supported by the National Science Foundation [OAC-1931363, ACI-1553685], the National Institute of Food and Agriculture [COL0-FACT-2019], and a Jack Cochran Family Professorship.

References

[1] M.-J. Kraak and F. Ormeling, *Cartography: visualization of geospatial data*. CRC Press, 2020.

[2] A. Mazher, "Visualization framework for high-dimensional spatio-temporal hydrological gridded datasets using machine-learning techniques," *Water*, vol. 12, no. 2, p. 590, 2020.

[3] N. Wang, T. W. Biggs, and A. Skupin, "Visualizing gridded time series data with self organizing maps: an application to multi-year snow dynamics in the northern hemisphere," *Computers, environment and urban systems*, vol. 39, pp. 107–120, 2013.

[4] J. Ahrens, B. Geveci, and C. Law, "Paraview: An end-user tool for large data visualization," *The visualization handbook*, vol. 717, no. 8, 2005.

[5] D. Weitzel, M. Zvada, I. Vukotic, R. Gardner, B. Bockelman, M. Rynge, E. F. Hernandez, B. Lin, and M. Selmeci, "Stashcache: a distributed caching federation for the open science grid," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*, 2019, pp. 1–7.

[6] D. Rammer, K. Bruhwiler, P. Khandelwal, S. Armstrong, S. Pallickara, and S. L. Pallickara, "Small is beautiful: Distributed orchestration of spatial deep learning workloads," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2020, pp. 101–111.

[7] S. Mitra, P. Khandelwal, S. Pallickara, and S. L. Pallickara, "Stash: Fast hierarchical aggregation queries for effective visual spatiotemporal explorations," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2019, pp. 1–11.

[8] M. U. Ahmed, R. A. Zaheer, and M. A. Qadir, "Intelligent cache management for data grid," in *Proceedings of the 2005 Australasian workshop on Grid computing and e-research-Volume 44*, 2005, pp. 5–12.

[9] S. Quinn and M. Gahegan, "A predictive model for frequently viewed tiles in a web map," *Transactions in GIS*, vol. 14, no. 2, pp. 193–216, 2010.

[10] G. Fox, G. Aydin, H. Bulut, H. Gadgil, S. Pallickara, M. Pierce, and W. Wu, "Management of real-time streaming data grid services," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 7, pp. 983–998, 2007.

[11] T. Buddhika, M. Malensek, S. L. Pallickara, and S. Pallickara, "Synopsis: A distributed sketch over voluminous spatiotemporal observational streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2552–2566, 2017.

[12] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," *arXiv preprint arXiv:1703.00395*, 2017.

[13] T. Dumas, A. Roumy, and C. Guillemot, "Autoencoder based image compression: can the learning be quantization independent?" in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 1188–1192.

[14] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Deep convolutional autoencoder-based lossy image compression," in *2018 Picture Coding Symposium (PCS)*. IEEE, 2018, pp. 253–257.

[15] Y. Choi, M. El-Khamy, and J. Lee, "Variable rate deep image compression with a conditional autoencoder," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3146–3154.

[16] X. Lu, Y. Tsao, S. Matsuda, and C. Hori, "Speech enhancement based on deep denoising autoencoder." in *Interspeech*, vol. 2013, 2013, pp. 436–440.

[17] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," in *2016 IEEE 16th international conference on data mining workshops (ICDMW)*. IEEE, 2016, pp. 241–246.

[18] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings, 2012, pp. 37–49.

[19] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, 2016.

[20] K. Bruhwiler, P. Khandelwal, D. Rammer, S. Armstrong, S. L. Pallickara, and S. Pallickara, "Lightweight, embeddings based storage and model construction over satellite data collections," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 246–255.

[21] A. Bulat and G. Tzimiropoulos, "Super-fan: Integrated facial landmark localization and super-resolution of real-world low resolution faces in arbitrary poses with gans," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 109–117.

[22] X. Xu, "Image interpolation based on the wavelet and fractal," *International Journal of Information Technology*, 2001.

[23] S. Mitra, D. Rammer, S. Pallickara, and S. L. Pallickara, "Glance: A generative approach to interactive visualization of voluminous satellite imagery," in *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 359–367.

[24] P. Khandelwal, D. Rammer, S. Pallickara, and S. L. Pallickara, "Mind the gap: Generating imputations for satellite data collections at myriad spatiotemporal scopes," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 92–102.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[26] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. Change Loy, "Esrgan: Enhanced super-resolution generative adversarial networks," in *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018, pp. 0–0.

[27] W. Cheng, M. Zhao, Z. Ye, and S. Gu, "Mfagan: A compression framework for memory-efficient on-device super-resolution gan," *arXiv preprint arXiv:2107.12679*, 2021.

[28] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, pp. 417–424.

[29] P. Khandelwal, S. Armstrong, A. Matin, S. Pallickara, and S. L. Pallickara, "Cloudnet: A deep learning approach for mitigating occlusions in landsat-8 imagery using data coalescence," in *2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE, 2022, pp. 117–127.

[30] A. Pondaven, M. Bakler, D. Guo, H. Hashim, M. Ignatov, and H. Zhu, "Convolutional neural processes for inpainting satellite images," *arXiv preprint arXiv:2205.12407*, 2022.

[31] Y. Liu and V. Caselles, "Exemplar-based image inpainting using multiscale graph cuts," *IEEE transactions on image processing*, vol. 22, no. 5, pp. 1699–1711, 2012.

[32] M. Bertalmio, L. Vese, G. Sapiro, and S. Osher, "Simultaneous structure and texture image inpainting," *IEEE transactions on image processing*, vol. 12, no. 8, pp. 882–889, 2003.

[33] U. Demir and G. Unal, "Patch-based image inpainting with generative adversarial networks," *arXiv preprint arXiv:1803.07422*, 2018.

[34] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative image inpainting with contextual attention," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5505–5514.

[35] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, "Image inpainting for irregular holes using partial convolutions," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 85–100.

[36] L. Liu and Y. Liu, "Load image inpainting: An improved u-net based load missing data recovery method," *Applied Energy*, vol. 327, p. 119988, 2022.

[37] C. Lab. Multivariate adaptive constructed analogs (maca). [Online]. Available: https://www.climatologylab.org/maca.html

[38] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. 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*, 2017, pp. 4681–4690.

[39] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[40] R. G. Deshpande, L. L. Ragha, and S. K. Sharma, "Video quality assessment through psnr estimation for different compression standards," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 11, no. 3, pp. 918–924, 2018.

[41] M. Grinberg, *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[42] Redis. [Online]. Available: https://redis.io/

[43] F. Yang, E. Tschetter, X. Léauté, N. Ray, G. Merlino, and D. Ganguli, "Druid: A real-time analytical data store," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 157–168.

[44] Tensorflow. Optimize machine learning models. [Online]. Available: https://www.tensorflow.org/model_optimization

[45] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[46] Tensorflow. Tensorflow lite. [Online]. Available: https://www.tensorflow.org/lite

[47] J. K. Udupa, "Three-dimensional visualization and analysis methodologies: a current perspective," *Radiographics*, vol. 19, no. 3, pp. 783–806, 1999.