RmdnCache: Dual-Space Prefetching Neural Network for Large-Scale Volume Visualization

Jianxin Sun, Xinyan Xie, and Hongfeng Yu

Abstract— Volume visualization plays a significant role in revealing important intrinsic patterns of 3D scientific datasets. However, these datasets are often large, making it challenging for interactive visualization systems to deliver a seamless user experience because of high input latency that arises from I/O bottlenecks and limited fast memory resources with high miss rates. To address this issue, we have proposed a deep learning-based prefetching method called *RmdnCache*, which optimizes the data flow across the memory hierarchy to reduce the input latency of large-scale volume visualization. Our approach accurately prefetches the content of the next view to fast memory using learning-based prediction while rendering the current view. The proposed deep learning architecture consists of two networks, RNN and MDN in respective spaces, which work together to predict both the location and likelihood distribution of the next view for defining an optimal prefetching range. Our method outperforms existing state-of-the-art prefetching algorithms in reducing overall input latency for visualizing real-world large-scale volumetric datasets.

Index Terms—Large-scale data, volume visualization, deep learning, prefetching

1 Introduction

Interactive 3D visualization techniques help researchers across various domains, from scientific research to medical practices, effectively discover informative patterns and extract valuable insights from datasets. However, with the exponential growth in the size of 3D datasets, particularly those generated from high granularity simulations or extensive sensing techniques in scientific research, it has become increasingly difficult to visualize such large-scale datasets on hardware systems with comparably limited memory resources and I/O bandwidths. In addition, to ensure a seamless user experience, interactive 3D visualization systems require an upper bound on the input latency to update the 3D scene in response to a user's new point of view (POV).

Those issues are addressed by several main approaches such as caching [15], multithreading [29, 36, 51], prefetching [40], data compression [3,33,52], and multi-resolution modeling [23,24,26]. However, most existing studies aim to either provide a unified solution for a general memory system to improve memory latency or optimize a specific situation that may not be directly applicable to an interactive 3D visualization system with a complex architecture and multiple constraints. Jointly optimizing different system components to efficiently and effectively manage the data movement across a memory hierarchy becomes the key to realizing a responsive, interactive 3D visualization system for large-scale datasets.

Many research efforts have been made to improve the performance of an interactive visualization system. For example, application-aware data replacement policies are proposed to use a client-server architecture to efficiently move data from back-end to front-end by prefetching data ahead of user current exploration [5,41]. Those methods deal with diverse types of datasets using predefined learning models or heuristic methods that mainly rely on a priori knowledge. As a result, the prediction may not be accurate enough to prevent suboptimal prefetching decisions for a specific dataset. Although many traditional predictive models [17, 18, 30, 64] adopt the nature of user visual exploration patterns, such as locality of exploration and correlation among neighboring POVs, it still needs cumbersome manual parameter tuning to yield an optimal result for each type of dataset. Data-centric methods, especially deep learning, have emerged with great momentum in recent years. Artificial neural network (ANN) shows its efficacy as a universal approximator driven by the extensive collection of training datasets, which helps overcome its weakness of over-fitting as a unified learning model. Deep learning methods are used to investigate memory access patterns to improve the accuracy and efficiency of memory prefetching [10, 25]. However, the insights from those investigations are not significantly beneficial to a complex interactive 3D visualization system across multiple hardware and software stack layers. Although a deep learning-based method [28] is proposed to visualize 3D flow fields, the dynamics of user interaction in 3D space are missing.

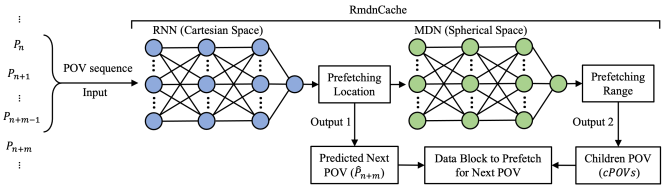

Fig. 1: Overview of the RmdnCache.

To tackle these challenges, we present *RmdnCache*, a deep learning-based prefetching neural network for interactive volume rendering of large-scale volume datasets using a multi-resolution framework. The core idea is to improve the performance of large-scale volume visualization applications by prefetching based on users' exploratory behaviors. The principal component of the algorithm is a novel deep neural network that predicts, during the prefetching phase, both the location of the next POV in a 3D Cartesian space and its prefetching range derived from an isocontour in a 2D spherical space. Both the predicted location and range will jointly determine a dynamic prefetching content with a lower miss rate for the next POV compared to other existing prefetching algorithms for multi-resolution frameworks. Fig. 1 shows an overview of the RmdnCache prefetching pipeline with its input (POVs sequence) and two outputs (prefetching location and range) for determining the data to prefetch. The training and testing datasets are POV trajectories collected from professional users for effective learning of user exploration patterns. Our solution can achieve a shorter caching time, the main contributor to the input latency, thereby improving the responsiveness of the interactive volume visualization system. To the best of our knowledge, this is the first work utilizing a deep learning predictive model to solve prefetching in a multi-resolution framework. The main contributions of this work include:

- A deep learning based prefetching algorithm to more accurately predict and prefetch the content of interest with a low cache miss rate and improved input latency, thereby enhancing the responsiveness of interactive visualization of large-scale datasets.

• *All authors are with the University of Nebraska-Lincoln. E-mail: {jianxin.sun, xinyan} @huskers.unl.edu, yu@cse.unl.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

- Analysis of how the hyperparameters of the network and the system configurations affect the overall performance.
- A training dataset of POV sequences collected from human users for training predictive models.

Even though our implementation of RmdnCache is tailored towards large-scale volume rendering, we anticipate that its fundamental principles and design philosophy can also be expanded to other 3D visualization applications.

2 Related Work

2.1 Out-of-core Methods

Out-of-core algorithms are extensively used in large-scale scientific data visualization, such as I/O-efficient volume rendering, isosurface computation, and streamline computation [4, 7, 38, 48]. By taking into account the distance between the camera view and each data segment in the current view, multi-resolution techniques [47, 53, 54] selectively load data segments with varying levels of detail [61], resulting in a smaller amount of data being loaded for rendering while still maintaining the similar level of rendering quality. Recent work utilizes neural networks [22, 57] to approximate complex shapes with various levels of detail. Utilizing an efficient disk data layout [42] or a pre-computed lookup table [11] can help efficiently access raw data in real-time visualization like progressive slicing and particle traces. Cox and Ellsworth [16] propose a framework for out-of-core scientific visualization systems by modifying the I/O subsystem based on application-controlled demand paging. The method utilizes the fact that many important visualization tasks only need to touch a small portion of large datasets at a time. Ueng et al. [46], Leutenegger and Ma [32] use a similar approach to perform on-demand loading of necessary data for visualization tasks by modifying the organization of actual data on disk by using octree or R-tree partition to handle structured and unstructured data. Researchers develop various out-of-core algorithms in isosurface generation and volume rendering from large-scale datasets. Chiang and Silva [13, 63] propose a solution for the stabbing problem or interval search problem. Sulatycke and Ghose [50] adapt the in-core data structure to an out-of-core setting with a multithreaded implementation. Bajaj et al. [2] propose a method using the extension of the seed-based technique for efficient isosurface extraction [1]. Sutton and Hansen [56] propose the T-BON (Temporal Branch-On-Need Octree) technique for fast extraction of isosurfaces of time-varying datasets. Farias and Silva [20] present a memory-insensitive approach and ZSWEEP algorithm [19] for the direct volume rendering of arbitrarily large unstructured grids on machines with limited memory. Our rendering framework utilizes an out-of-core method to dynamically load regions of interest, effectively managing large-scale datasets.

2.2 Prefetching

Prediction-based speculative prefetching [14, 21] can dramatically decrease the cache miss rate for interactive visualization applications, thereby reducing the overall input latency. Prefetching adds an extra layer of optimization on cache for the next POV, so out-of-core rendering methods that utilize both caching and prefetching generally give a lower miss rate than methods that only utilize caching [31]. The prefetching can also operate in parallel with rendering under modern multi-core systems to eliminate its overhead for faster overall rendering time. Battle et al. [5] present a prefetching method named ForeCache for interactive visualization. ForeCache uses the Markov model to predict future data requests, and its prediction accuracy becomes the bottleneck for improving the effectiveness of prefetching. Yu et al. [64] propose an application-aware algorithm (APPA) that makes predictions according to the importance of the data. However, domain knowledge is needed for a specific dataset to calculate the importance of the data block for constructing the lookup table. A data-driven model, such as deep neural network, is an emerging method of constructing a predictive model from data. However, there are sporadic research works targeting prefetching using deep learning methods for large-scale volume visualization [60]. Hong et al. [28] propose a prefetching method using Long Short-Term Memory, LSTM, to model data access patterns for particle tracing in flow field visualization. The model is designed only for exploring the flow field for the training data are sampled pathlines in its domain, so it does not apply to general 3D volumetric data visualization. The primary contribution of our work is the introduction of a more effective prefetching method that leverages deep learning to account for users' exploratory behaviors.

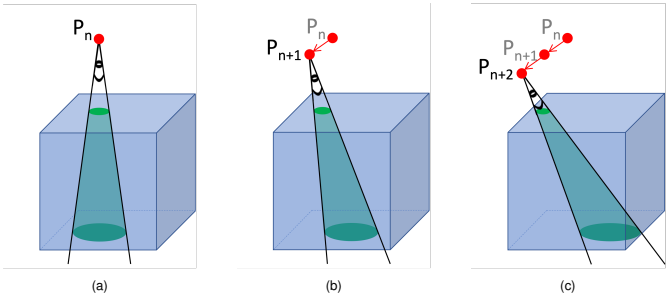

Fig. 2: Interactive 3D visualization mechanism. (a), (b) and (c) demonstrate three sequential view-dependent operations by a user. The blue cube is the volume dataset for visualization. Green volume is the visible content to render for the current POV. Each red vector connects contiguous POVs. All red vectors are connected in order to form a 3D trajectory of data exploration of the user.

2.3 Implicit Neural Representation

Deep learning-based rendering synthesis [6, 35] has emerged in recent years to improve the performance of rendering volumetric datasets across spatial and temporal domains. The implicit neural representation [45, 49] is trained as a more efficient and effective model to leverage the GPU acceleration during inference. However, rendering synthesis methods need to first create comprehensive input and label pairs as training datasets considering extensive combinations between view and data-dependent operations. For large-scale data, preparing such training datasets itself takes a considerable amount of computational resources. Moreover, the training time is also generally long to achieve an accurate inferencing result. The memory footprint of rendering synthesis is normally large to generate the resulting image in on shot. Compressing methods leveraging neural representation [37, 58] are proposed to decrease the network size and optimize I/O intensive operations. Although random access is supported to query value or gradient directly from the neural representation without decompression, its training time is also inevitably long using powerful GPU when handling complex large-scale scientific datasets. The rendering frameworks we aim to optimize are traditional multi-resolution-based renderers, where the compression is applied only to data partitions without extensive training or modeling toward the entire dataset.

3 Methods

3.1 Problem Formulation

Interactive large-scale volume rendering is a real-time 3D rendering process on selected data retrieved according to queries made by user operations. User operations can be classified into two main groups, which are data-dependent operations (e.g., applying mapping or transfer function on data) and view-dependent operations (e.g., applying camera movement). In this work, we focus on optimizing the main view-dependent operations, specifically the zooming, rotating, and panning operations. Fig. 2 demonstrates a typical interactive 3D visualization as a user triggers view-dependent operations in sequence, where P_n indicates a POV and n represents the sequential index of the POV as the user explores the 3D volume. This work investigates prefetching algorithms under a commonly used GPU-accelerated multi-resolution framework for interactive 3D volume rendering of large-scale volumetric data, where units of data moving across memory hierarchy are partitioned data blocks, or *microblocks*, compressed with various levels of detail. Lower resolution content is used at further regions from the camera to decrease the overall data size for caching and rendering, the spatial size

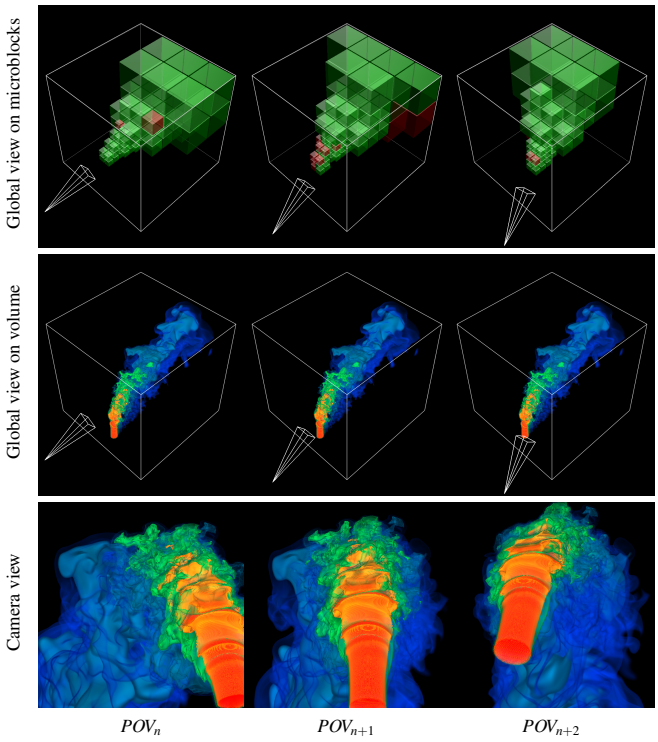

Fig. 3: Various views of a multi-resolution volume rendering on the Flame dataset. The global view on microblocks (first row) shows visible microblocks of 4 resolution levels covered by the current POV. The white prism indicates the location and orientation of the current POV. Larger blocks (further from the POV) represent microblocks with lower resolutions, while smaller blocks (closer to the POV) represent microblocks with higher resolutions. Red blocks and green blocks represent miss and hit microblocks in the cache for the current POV. Global view on volume (second row) shows the spatial relationship between POV and the volume-rendered data. The camera view (third row) shows the final volume rendering result for the specific POV.

of the unit block scales based on the distance between the POV and the centroid of the block for an appropriate level of detail [24, 44]. Fig. 3 demonstrates an example of multi-resolution rendering of three POVs on a Flame dataset.

Fig. 4 outlines the structure of a typical GPU-accelerated multi-resolution volume rendering pipeline with prefetching. Please refer to the appendix for the detailed algorithm. First, visible blocks from the current POV are retrieved based on the spatial relationship between the camera and microblocks of all resolution levels. Second, the system cache is updated by loading those visible blocks and following certain cache replacement policies. Least Recently Used (LRU) algorithm is utilized in this work for generalization. Third, the visible microblocks will be copied to GPU memory for rendering specific visualization results. Since the size of the data copied to GPU for rendering using the multi-resolution method is much smaller compared to the raw data size, according to our measurement, the memory copying time from RAM and VRAM is negligible compared to the time used on other procedures. Detailed measurements will be given in Sec. 4.6.2. Finally, prefetching predicts the visible microblocks for the next POV and updates the cache in parallel with rendering. In order to minimize the overall input latency, prefetching must stop, even if it is not finished, at the moment when the rendering is done. For the POV of sequence n, the sum of caching time (TC_n) and rendering time (TR_n) determines the total input latency, the total time that elapses from the user action to the finish of rendering. The rendering time is determined by the complexity of the rendering algorithm and the capability of rendering hardware, which is out of the scope of this paper. Our objective is to optimize the prefetching at the current POV to decrease the miss rate of caching at the next POV. A lower miss rate results in less total I/O overhead with less data

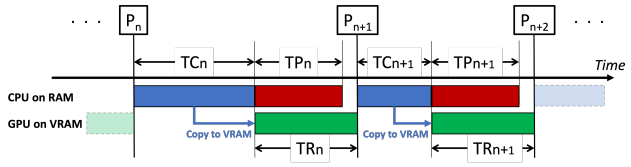

Fig. 4: Interactive 3D visualization pipeline with continuous view-dependent operations. P_n is the nth POV. TC_n, TP_n, and TR_n are the times used for caching, prefetching, and rendering of the nth POV, respectively.

movement and reduces the caching time for better responsiveness of the system. For a current POV (P_n), the objective function of our method is to minimize the miss rate at the next POV (P_{n+1}). The miss rate can be expressed as:

$$R_{miss}(P_{n+1}) = \frac{|\overline{B_{cached}(P_{n+1})} \cap B_{visible}(P_{n+1})|}{|B_{visible}(P_{n+1})|} \quad (1)$$

where

$$B_{cached}(P_{n+1}) = C\Big(C\big(B_{cached}(P_n),\ B_{visible}(P_n)\big),\ B_{prefetch}(P_n)\Big) \quad (2)$$

B denotes a set of microblocks. Function $B_{visible}(P_n)$ retrieves a set of visible microblocks requested for a specific POV. $B_{prefetch}(P_n)$ predicts a set of visible microblocks for the next POV P_{n+1} given the current POV P_n. $B_{cache}(P_n)$ represents currently cached microblocks of a POV. The function $C(cache,\ new\ blocks\ to\ cache)$ carries out caching by inserting missing new microblocks into the cache through replacement and then returning all the microblocks in the cache. The operator $|\cdot|$ returns the cardinality of a set. The function $R_{miss}(P_n)$ computes the miss rate, which is the ratio between the number of missing microblocks and the total number of visible microblocks. The goal is to design the function of $B_{prefetch}(P_n)$ that can accurately and efficiently predict visible blocks as close as $B_{visible}(P_{n+1})$. This prediction is computed on every view-dependent operation from a user in real time.

3.2 Proposed Solution

3.2.1 Rationale

If we can accurately predict the location of P_{n+1} from previous m POVs, $\{P_{n-m+1},\ P_{n-m+2}, \cdots,\ P_n\}$, we can simply construct the prediction function for prefetching as:

$$B_{prefetch}(P_{n+1}) = B_{visible}\big(\mathscr{H}(P_{n-m+1},\ P_{n-m+2},\ \cdots,\ P_n)\big) \quad (3)$$

where $\mathscr{H}(P_{n-m+1},\ P_{n-m+2}, \cdots,\ P_n)$ is a trajectory prediction function that predicts a POV ($\hat{P}_{n+1}$) having the shortest Euclidean distance to the next POV (P_{n+1}). We call this type of prefetching as *point prefetching*. Intuitively, the point prediction can be simply addressed using a typical seq2seq [55] recurrent neural network (RNN) [34] model, such as the long short-term memory (LSTM) model [27], to implement the function $\mathscr{H}$.

Nonetheless, when dealing with new user input, there will always be a discrepancy between the prediction and the ground truth. Using Eq. (3) for point prefetching based on the predicted POV position will likely fail to encompass all visible microblocks in the subsequent POV, as shown in Fig. 5a. One solution is to use the *range prefetching* to cover a broader range of microblocks, as shown in Fig. 5b. However, the size of the prefetching range has to be justified carefully: If too small, it fails to cover targeting microblocks; if too large, it will prefetch too many microblocks with weak locality, and as a result, the cached microblocks will be flushed out by uncorrelated microblocks, which will inevitably increase the miss rate of subsequent caching.

Based on this observation, we propose a new algorithm for predicting visible microblocks for the next POV. The algorithm still starts from a POV location prediction but then adds a range determination on top of it. The location provides a center point with the highest probability for

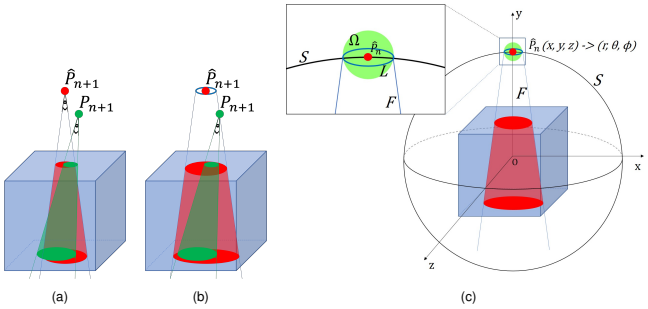

Fig. 5: (a) and (b) show point prefetching and range prefetching, respectively. $\hat{P}_{n+1}$ is the predicted POV location while P_{n+1} is the ground truth. The green region is the visible region of P_{n+1}. Red regions in (a) and (b) represent the visible region by point prefetching and range prefetching of $\hat{P}_{n+1}$, respectively. (c) shows how to simplify mixture density prediction Ω in 3D Cartesian space as L in 2D spherical space.

successful prefetching, while the range provides a statistically sound evaluation of how far the prefetching should extend to mitigate the prediction error.

3.2.2 Prefetching Location Prediction

We apply a data-driven method using deep learning models to predict prefetching location by learning features from exploration data collected from real users. We implement an interactive visualization tool to collect training data from human users. The software uses traditional 3D volume rendering, allows users to explore a dataset through view-dependent operation interactively, and meanwhile collects the sequence of POVs of the user's explorations, where the 3D location (x,y,z) of each POV in the Cartesian coordinate is recorded. We use these sequences to train an LSTM model as RNN to predict the location of the next POV from unseen trajectories in the inferencing phase. Long Short-Term Memory (LSTM) [65] is a type of recurrent neural network (RNN) architecture that has proven effective in modeling sequential data. Unlike traditional RNNs, LSTM networks are designed to mitigate the vanishing gradient problem, allowing them to maintain and propagate information over extended sequences. LSTMs are widely used in various fields, including natural language processing, speech recognition, time series forecasting, and more.

3.2.3 Prefetching Range Determination

Although RNN can predict the next POV location from previous POVs, it does not provide rigorous statistical confidence in the prediction. Therefore, we utilize the idea of a mixture density network (MDN) [8] to determine an appropriate prefetching range for each predicted POV from LSTM. A Mixture Density Network (MDN) is a neural network architecture used for modeling complex probability distributions. Unlike traditional neural networks that output deterministic values, an MDN generates a mixture of multiple probability distributions with respective weights for importance. This allows MDNs to capture multimodal distributions and handle uncertainty in predictions. MDN can predict a distribution rather than a single result by learning from the data. In this work, we select multivariate Gaussian distribution as the component of the mixture distribution for its capability of modeling arbitrary probability density functions.

When training an MDN model, the input of the MDN network is the output of the RNN network, and the label of the MDN network is the ground truth of a specific POV location. The trained MDN model can predict a 3D region Ω with a certain likelihood surrounding its predicted mean, which can be seen as a predicted POV $\hat{P}_n$, as illustrated as the green region in Fig. 5c. Thus, the predicted prefetching range is similar to a conical frustum F enclosing Ω. The visible microblocks can be calculated by projecting F on the data volume, as shown in the red region in Fig. 5c. It is computationally expensive to predict Ω in 3D using MDN. This will greatly decrease the prefetching efficiency and result in higher input latency due to slow inferencing. Assuming that

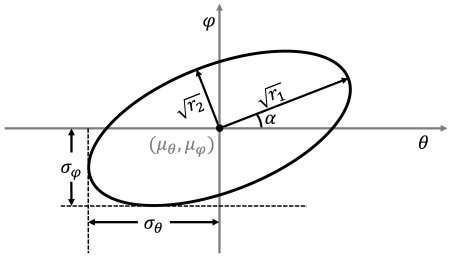

Fig. 6: Example of an isocontour from a bivariate Gaussian distribution with parameters that define its shape and orientation.

$\hat{P}_n$ is located on a spherical surface S, we can see that the intersection of F and S is a 2D contour L, which is all it needed to retrieve the prefetching range. Based on this observation, we manage to reduce the computational cost of MDN by predicting a bivariate Gaussian mixture in 2D rather than a trivariate Gaussian mixture in 3D. We first convert the RNN output, in a 3D Cartesian coordinate (x,y,z), to a spherical coordinate (r,θ,φ). The prefetching range, in spherical coordinates, can be retrieved by referencing the joint distribution of the polar angle (θ) and the azimuthal angle (φ). This joint distribution can be directly learned and predicted by a bivariate Gaussian mixture density network on the sphere surface S with the corresponding radius r.

We train the MDN model by fitting bivariate Gaussian functions across θ and φ, and directly reuse r from the output of RNN. The prefetching range can be extracted by solving an isocontour generation problem on the mixture density. The isocontours are lines or curves that connect points of equal value on the surface of the Gaussian mixture probability density function. It is nontrivial to solve this problem, especially when we have a large number of Gaussian components for the mixture and their weights are close to a uniform distribution. By analyzing the prediction of the trained MDN model on the validation dataset, we observed that, among a given group of predicted Gaussian components, there is always a dominant component with a much larger weight compared to the rest of the components most of the time. The details of the observation will be discussed in Sec. 4.5. Therefore, we can approximate the mixture distribution using the single Gaussian component with the highest weight. In this way, we can once again save a considerable amount of processing time for extracting the prefetching range by only examining a single bivariate Gaussian distribution. The prefetching range is the isocontour or the level set of the dominant bivariate Gaussian distribution, and its shape is an ellipse. For a given covariance matrix of the bivariate Gaussian distribution on θ and φ:

$$X(\theta,\varphi) = \begin{bmatrix} \sigma_\theta^2 & \rho\cdot\sigma_\theta\cdot\sigma_\varphi \\ \rho\cdot\sigma_\theta\cdot\sigma_\varphi & \sigma_\varphi^2 \end{bmatrix} \tag{4}$$

where σ_θ and σ_φ are the standard deviation of θ and φ respectively, while ρ is the correlation coefficient of θ and φ. The parameters that define the isocontour for $\theta \in [\mu_\theta - \sigma_\theta, \mu_\theta + \sigma_\theta]$ and $\varphi \in [\mu_\varphi - \sigma_\varphi, \mu_\varphi + \sigma_\varphi]$ as shown in Fig. 6 can be calculated as:

$$r_1 = \frac{\sigma_\theta^2+\sigma_\varphi^2}{2} + \sqrt{(\frac{\sigma_\theta^2-\sigma_\varphi^2}{2})^2 + (\rho\cdot\sigma_\theta\cdot\sigma_\varphi)^2} \tag{5}$$

$$r_2 = \frac{\sigma_\theta^2+\sigma_\varphi^2}{2} + \sqrt{(\frac{\sigma_\theta^2-\sigma_\varphi^2}{2})^2 - (\rho\cdot\sigma_\theta\cdot\sigma_\varphi)^2} \tag{6}$$

$$\alpha = \begin{cases} 0 & \text{if } \rho = 0 \text{ and } \sigma_\theta^2 \geq \sigma_\varphi^2 \\ \frac{\pi}{2} & \text{if } \rho = 0 \text{ and } \sigma_\theta^2 < \sigma_\varphi^2 \\ arctan(\frac{\rho\cdot\sigma_\theta\cdot\sigma_\varphi}{r_1-\sigma_\theta^2}) & else \end{cases} \tag{7}$$

The parametric equation of the isocontour in spherical space using the independent variable ω can be expressed as:

$$\begin{aligned} \theta(\omega) &= \sqrt{r_1}\cdot cos(\alpha)\cdot cos(\omega) - \sqrt{r_2}\cdot sin(\alpha)\cdot sin(\omega) + \mu_\theta \\ \varphi(\omega) &= \sqrt{r_1}\cdot sin(\alpha)\cdot cos(\omega) + \sqrt{r_2}\cdot cos(\alpha)\cdot sin(\omega) + \mu_\varphi \end{aligned} \tag{8}$$

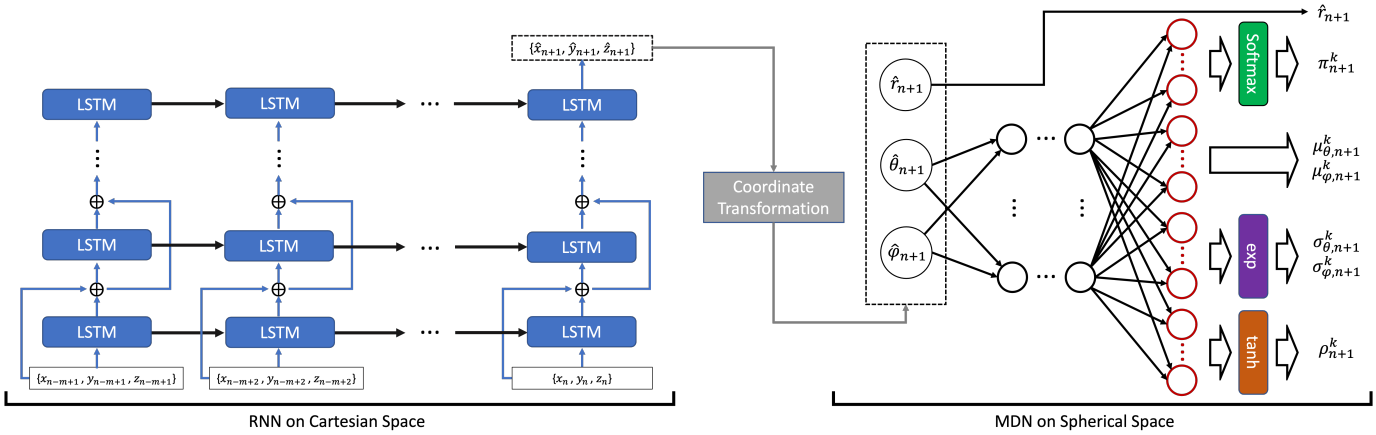

Fig. 7: Network architecture of the proposed RmdnCache. The RNN operates in Cartesian space, while the MDN operates in Spherical space.

The parametric equation of the isocontour, the prefetching range, in the Cartesian space can be transformed as:

$$\begin{aligned} x(\omega) &= r \cdot sin[\theta(\omega)] \cdot cos[\varphi(\omega)] \\ y(\omega) &= r \cdot sin[\theta(\omega)] \cdot sin[\varphi(\omega)] \\ z(\omega) &= r \cdot cos[\theta(\omega)] \end{aligned} \tag{9}$$

where r is the magnitude of the POV in spherical space after converting the predicted result by RNN in Cartesian space.

3.3 Proposed Network Architecture

We propose RmdnCache–a recursive mixture density network for prefetching–that consists of two deep learning networks, RNN and MDN, operating in the Cartesian space and the spherical space, respectively, as shown in Fig. 7. The LSTM is selected as the RNN model and designed with adequate features in hidden layers for better performance. For the current POV (P_n), the LSTM network takes the previous m POVs as the input sequence. Each element of the sequence is a three-dimensional vector, $\{x,\ y,\ z\}$, in the Cartesian space. Only the last output from the final LSTM cell in the network is used as the prediction for the next POV ($\hat{P}_{n+1}$). Since LSTM is solving a regression problem, the mean square error (MSE) is used as the loss function:

$$Loss_{MSE} = \frac{\sum_{n=1}^{N} L2Norm(P_n,\ \hat{P}_n)^2}{N} \tag{10}$$

where $L2Norm()$ is the function to calculate the L2-norm between the predicted POV ($\hat{P}_n$) and the ground truth POV (P_n). N is the total number of input sequences.

Once the LSTM network is trained, we fix its parameters and train the MDN network on top of it in a transfer-learning fashion. This predicted location will be transformed into the spherical space for the MDN network by using the following equations:

$$r = \sqrt{x^2+y^2+z^2};\ \theta = arctan\frac{\sqrt{x^2+y^2}}{z};\ \varphi = arctan(\frac{y}{x}) \tag{11}$$

where

$$r \geq 0;\ 0° \geq \theta \geq 180°;\ 0° \geq \varphi \geq 360°.$$

The MDN model only takes θ and φ as input and outputs a set of parameters that define the predicted bivariate Gaussian mixture distribution. We denote this parameter set as $\hat{y}$:

$$\hat{y} = \{\bigcup_{k=1}^{K} \pi^k,\ \bigcup_{k=1}^{K} \mu^k,\ \bigcup_{k=1}^{K} \sigma^k,\ \bigcup_{k=1}^{K} \rho^k\} \tag{12}$$

where K is the number of Gaussian components used for the mixture. π^k represents the weight for the kth component. μ^k, σ^k, and ρ^k are the mean, the standard deviation, and the correlation coefficient for kth bivariate Gaussian component, respectively. The output logits of MDN need to be consistent with the range of valid weights (sum of weights is 1.0), standard deviations (greater than 0), and correlation coefficients (between -1 to 1). This can be satisfied using the following functions:

$$\pi^k = Softmax(\pi_{logits}), \sigma^k = exp(\sigma_{logits}), \rho^k = tanh(\rho_{logits}) \tag{13}$$

The probability density function of K Gaussian mixture is:

$$P(\theta,\varphi) = \sum_{k=1}^{K} \pi^k \cdot G(\theta,\varphi \mid \mu^k,\ \sigma^k,\ \rho^k) \tag{14}$$

where G is a bivariate Gaussian distribution on θ and φ:

$$\begin{aligned} G(\theta,\varphi \mid \mu^k,\ \sigma^k,\ \rho^k) &= \frac{1}{2\pi\sigma_\theta\sigma_\varphi\sqrt{1-\rho^2}} exp[-\frac{z}{2(1-\rho^2)}] \\ z &= \frac{(\theta-\mu_\theta)^2}{\sigma_\theta^2} - \frac{2\rho(\theta-\mu_\theta)(\varphi-\mu_\varphi)}{\sigma_\theta\sigma_\varphi} + \frac{(\varphi-\mu_\varphi)^2}{\sigma_\varphi^2} \end{aligned} \tag{15}$$

The goal of the MDN network is to maximize the weighted sum of likelihood from each component for a given input, equivalently, to minimize negative log operation on top of it. The loss function of MDN can be written as:

$$Loss = \sum_{n=1}^{N} -log[\sum_{k=1}^{K} \pi_n^k \cdot G(\theta_n,\varphi_n \mid \mu_n^k,\ \sigma_n^k,\ \rho_n^k)] \tag{16}$$

The proposed predictive model through deep learning tries to leverage users' exploratory behaviors to improve the effectiveness of prefetching. Once the model is trained, we can use the method mentioned in Sec. 3.2.3 to retrieve a prefetching range for each predicted POV.

3.4 Visible Blocks Retrieval

The visible blocks need to be retrieved based on the prefetching range given by the network. In theory, the ideal visible microblocks need to fill in the region covered by the prefetching range with a single ellipse shape as shown in the red region of Fig. 8a. However, it is computationally expensive to exactly compute range coverage for an ellipse with a high degree of rotation and radius freedom of the major and the minor axes. We propose a solution to approximate such range coverage using the union of multiple point coverages as shown in Fig. 8b. We compute each point coverage starting from points $(p_0, p_1, ...)$ uniformly distributed on the edge of the elliptic prefetching contour by calculating the microblocks in the intersection between the dataset and the standard cone shape regions projected from those points. We call those sampled points child POVs (cPOVs). The final bag of visible blocks is the union of all the point coverage from those cPOVs and the center point of the

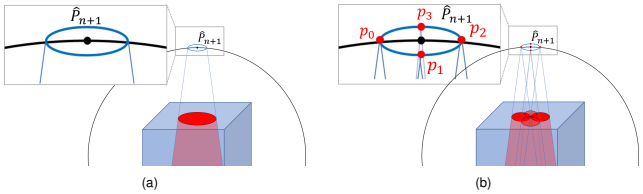

Fig. 8: Visible block retrieval. (a) is the ideal method using range coverage but with high computational complexity. (b) is the proposed faster method by aggregating multiple point coverages. The four red sampled points on the blue prefetching contour are cPOVs.

contour. The finer sampled cPOVs used, the closer the block union is to the real range coverage of the prefetching contour. It is easy to compute the point coverage of a cPOV by itself or leverage a lookup table to decrease the retrieval time further. Although more cPOVs make the union of point coverages from those cPOVs closer to the original range coverage, the longer visible microblocks searching time during the union operation is detrimental to prefetching efficiency. The sampling density of cPOV can be tuned by trading off the prefetching accuracy and efficiency.

4 Experiments and Evaluation

In order to extensively examine the prefetching performance of an interactive visualization system on large-scale datasets, we implement a general multithreaded volume renderer utilizing a multi-resolution framework with GPU acceleration by following the pipeline of Fig. 4. The framework we implemented is general enough to capture common procedures of a typical GPU-accelerated multi-resolution 3D visualization system. This framework is also used in collecting training data from users. The details of the renderer can be found in section 2 of the appendix.

4.1 Considered Prefetching Algorithms

To evaluate the performance of our RmdnCache, we examine three other previous works of prefetching methods, APPA [64], ForeCache [5] and LSTM [28], that we find applicable to multi-resolution visualization systems for handling large-scale datasets. APPA derives a parent POV (pPOV), which is a new POV further from the volume object by increasing the distance between the current POV and the centroid of the 3D volume, hoping that the larger range of visible microblocks covered by the pPOV derived from current POV will also cover most visible microblocks of the next POV. APPA actually realizes a range coverage retrieval through the point coverage of the pPOV. Both ForeCache and LSTM predict the next POV from a sequence of previous POVs using the Markov model and the deep learning RNN, respectively. The LSTM method we use here is similar to the LSTM used in Hong et al. [28] except replacing the input from particle trajectories of the flow field data to user trajectories for prediction of the next POV. Our RmdnCache utilizes both RNN and MDN to predict not only the next POV but also a series of cPOVs describing the accurate prefetching range. We evaluate RmdnCache against these existing algorithms with the key system-level metrics that determines the overall performance of the visualization system, including caching miss rate (MR), caching time (TC), rendering time (TR), and prefetching time (TP), which are main contributors to the overall input latency.

4.2 Datasets

Methods in the automatic selection of representative views and exploration paths [9, 59] can provide POVs based on the data-dependent evaluations for visualization systems without user interaction. Because our contribution is to improve the visualization systems with heavy user interaction, POVs from real human users need to be collected. The training trajectory datasets are collected from real users when exploring the volume dataset on our interactive 3D visualization framework. The format of the training dataset collected is trajectories of POVs in the Cartesian space. Multiple volume datasets with distinct spatial features and transfer functions are used for collecting comprehensive training datasets of trajectories. The users are scientists or researchers in computer science, including professors and graduate students. In order to improve the representativeness of the collected data, the users we selected either have previous experience in research in visualization or have an interest in large-scale volume visualization. All participants are required to sign the consent document beforehand. The visualization task involves exploring regions of interest for each user through a series of view-dependent operations. There are no restrictions on the user's exploratory behaviors or the time taken to complete the task. The initial point is randomly selected for each user. Collected training trajectories are cleaned by removing POV outliers due to users' misoperations to improve uniformity. The distribution of the training trajectories is balanced as they span the whole domain containing the volume rather than only focusing on a small region of interest. Examples of training trajectories can be found in the appendix. The datasets used to train the model are sequence and label pairs cut from the 50 training trajectories (340 POVs per trajectory) by applying a sliding window. The training and validation datasets are split from all training pairs with a ratio of 5 : 1. For evaluation, we test the performance of our model on unseen user trajectories exploring 4 unseen large-scale volume datasets with distinct spatial features and transfer functions. The large-scale volumetric datasets for testing are 1) Flame (6.45GB), 2) Truss (6.45GB), 3) Chameleon (4.23GB), and 4) Rayleigh-Taylor (4.01GB). Each testing dataset has a corresponding exploratory testing trajectory containing 400 POVs. Four resolution levels are used in the experiments by following the octree partition strategy to construct microblocks for multi-resolution visualization. Three evenly distributed distance thresholds from a POV are used to differentiate the microblocks of 4 levels of resolutions. The final volume rendering image resolution is 1024×1024. Please refer to the appendix for detailed information on the datasets and their microblock partitions.

4.3 Experimental Setup

A two-level memory hierarchy model, from storage to RAM, is used in the experiment for generalization. However, the benefit of prefetching can be easily scaled to multi-level memory hierarchies. The cache memory size is fixed as 200 microblocks for the multi-resolution rendering pipeline. Although the number of microblocks in the cache is fixed, the actual cache size is determined by the microblock size resulting from the partition of a specific dataset. We use a slow 4TB hard disk drive (HDD) with 5400 RPM on SATA interface as the storage to better evaluate the effectiveness of all the considered prefetching algorithms under such a worst-case scenario where high data movement latency between storage and RAM becomes the main bottleneck of the responsiveness. We also test the same experiments on a solid-state drive (SSD) with the SATA interface for its popularity nowadays. In addition, we use a lower-end NVIDIA GTX 1050 Ti GPU with a VRAM of only 4GB, which is insufficient for loading the entire volume of data to generate the interactive visualization. The hardware platform is a desktop with Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz with 8GB DDR4 DRAM 3200MHz. Our approach ensures a small memory footprint by dynamically fetching the required microblocks into memory based on user explorations. The operating system on the desktop is Ubuntu 20.04.4 LTS. We disable the system cache of the operating system so that our measurements of caching performance, dedicated to the interactive visualization task, are more faithful for an accurate evaluation.

4.4 Network Configuration and Training

The RNN network is an LSTM [27] designed with one hidden layer with 100 features. As LSTM operates on a sequence of Cartesian space coordinates, both the input and output dimensions consist of three components (x, y, z). The standard multi-layered unidirectional LSTM architecture is used. The stochastic gradient descent (SGD) optimizer is used with a learning rate set as 0.1. The raw dataset is formatted into sequence-to-sequence pairs for training the LSTM. The MDN network is basically built from fully connected MLP [43] and designed using one hidden layer with 20 features. The input consists of θ and φ in the

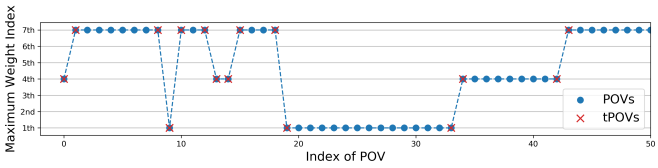

Fig. 9: Index of Gaussian components with the maximum weight across the first 50 POVs of a user's trajectory of the validation dataset from the prediction of MDN model with 7 Gaussian components. The red cross denotes transition POV (tPOV), where the maximum weight of the mixture shifts from one Gaussian component to another. The blue dots represent the POVs where there is a single dominant Gaussian component on weight without weight shift.

2D spherical space, while the output dimension is six times the number of Gaussian components for the mixture, where six is the total number of parameters for defining a Gaussian component (π, μ_θ, μ_φ, σ_θ, σ_φ, ρ). Adam optimizer is used with a dynamically adjusted learning rate. Once the LSTM network is trained, we freeze its parameters and use the practice of transfer learning to train the MDN network on top of LSTM. The rectified linear unit (ReLU) activation function is used in both LSTM and MDN networks.

The proposed deep learning networks are trained using the PyTorch software stack to accelerate the training and inferencing performance on a single NVIDIA GTX 1080Ti GPU. A validation dataset is selected to detect the overfitting on the training dataset and freeze the network parameters from updating through the early stop mechanism. The training of LSTM and MDN stop at around the 3000th and 150th epoch, respectively, before overfitting happens. The training time is around 90 minutes for LSTM and 20 minutes for MDN.

4.5 Hyperparameter Tuning

In order to optimize the network architecture for more accurate prediction to improve the prefetching efficacy, we evaluate and optimize the key parameters of the network through hyperparameter tuning. Detailed results of error using different hyperparameters of RNN window size and MDN mixture components number can be found in the appendix.

RNN Window Size We evaluate the RNN network performance by adjusting the input sequence length from 1 to 10 and find optimal window size of 3 gives the lowest MSE loss on the validation dataset, and thus it will be used as the optimal parameter for the LSTM part of our proposed network. After partitioning the training trajectories with a window size of 3, there are in total 16850 sequences and label pairs for training.

MDN Mixture Components Number We also evaluate the number of bivariate Gaussian components used for the MDN network. We observe the fact that increasing the number of components does not necessarily result in better performance. However, using more components will linearly improve the number of features in the logits layer, which enlarges the network's parameter space and consequentially increases the inferencing latency. Moreover, more labeled data are needed to train a more complex model properly to prevent the network from overfitting; otherwise, the generalization ability of the network will be diminished. Based on our evaluation, we pick 5 as the optimal number of components in our developed MDN network.

Dominant Gaussian Component Selection We also observe that, among all predicted Gaussian components, for most of the time, there is only one dominant Gaussian component that has the highest weight compared to other components. This can be proved by counting the frequency of the POV whose maximum weighted component changes to a different one compared to its previous POV, and we call such POV as transition POV (tPOV). A maximum weight transition only happens when more than one components have relatively close magnitudes of their weights, which means all those components with similar weight need to be considered to construct the final Gaussian mixture for an accurate prefetching range prediction. As shown in Fig. 9 from our experiment, as the current POV moves along the trajectory of all POVs, the tPOVs only appear with a very low frequency. We measure the frequency of tPOV among all POVs on our validation dataset under MDN models with a different number of Gaussian components as shown in Tab. 1. The results indicate that the tPOV frequency is below 9% for most MDN models, regardless of the number of Gaussian components used. Consequently, at least 91% of the time, we can select the single dominant Gaussian component with the highest weight for the final bivariate likelihood prediction, instead of aggregating all components. This approach speeds up MDN inference time with only a marginal loss in accuracy. For our RmdnCache model using 5 Gaussian components based on the hyperparameter tuning, there is no tPOV at all, which means there is no transition between components for maximum weight. This is because, for the distribution of validation datasets, the MDN using 5 Gaussian components can always catch the maximum likelihood of the underlying distribution to the same components. In summary, picking a Gaussian component number in the middle range is suitable for balancing the accuracy and performance. This means RmdnCache sacrifices even less accuracy from such optimization.

Table 1: Percentage of the tPOVs in all POVs of the trajectory under MDN models with different numbers of Gaussian components.

Components Number	2	3	4	5	6	7	8	9	10
Percentage	0%	4.5%	5.5%	0%	8.1%	8.6%	6.8%	5.6%	0%

4.6 Results

We train our proposed RmdnCache network with optimal hyperparameters on our training dataset and infer the pre-trained model in real time to retrieve potentially visible microblocks for the next POV. We run experiments on the interactive multi-resolution volume rendering framework using various prefetching algorithms on 4 unseen user trajectories exploring 4 unseen large-scale volumes. Fig. 10 shows the 4 testing volumes and their user exploratory trajectories, together with the detailed color and opacity transfer functions used for rendering. We also run the experiment without prefetching as a baseline where only LRU caching is utilized. The proposed RmdnCache uses the predicted next POV and 4 sampled cPOVs on its prefetching range to retrieve the visible microblocks. We measure the cache miss rate of each POV and its average during caching to evaluate the accuracy of predicting visible microblocks using different prefetching algorithms. The input latency of each POV and its average is measured to evaluate the responsiveness of the interactive visualization system using different prefetching algorithms. The final measurement is the average of a total of 10 repeated measurements with the same setting.

Fig. 11 demonstrates the spatial comparison between the ground truth user trajectory and the predicted trajectory from RmdnCache. It can be observed that the predicted trajectory aligns with the ground truth for most of the time except for some POVs deviating from the trajectory. However, the error in prefetching caused by such deviated POVs can be compensated by the prefetching range. This can be seen in the zoom-in views in both Cartesian and Spherical spaces as shown in Fig. 12 and Fig. 13. For each predicted POV, its ellipse-shaped prefetching range tries to cover the ground truth POV, so that the final prefetching region determined by RmdnCache, as mentioned in Sec. 3.2.3 and Sec. 3.4, will also cover most of the prefetching region derived from the ground truth POV. This is the main reason why RmdnCache performs better than other prefetching algorithms predicting a single POV.

4.6.1 Miss Rate

Fig. 14 shows the measured miss rate of the first 101 POVs on testing dataset 1 using the four considered prefetching algorithms and the LRU without prefetching. The detailed miss rate of all POVs for each testing dataset can be found in the appendix. In general, the miss rate of the first POV is always 1.0 because the cache is empty at this moment, and all visible microblocks needed to render this POV are missing. As a user explores the dataset through POV movement, new microblocks are needed for rendering the view of the current POV, and the cache is updated accordingly by loading the missing microblocks. Fig. 15 shows the cache is getting filled with new microblocks as the user

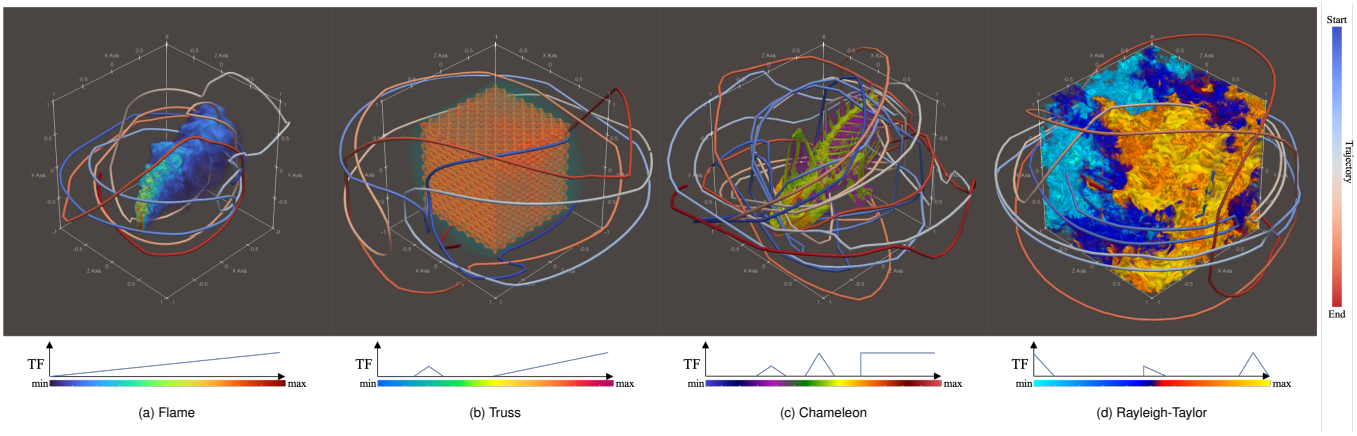

Fig. 10: Four large-scale testing volumetric datasets and their user exploratory trajectories.

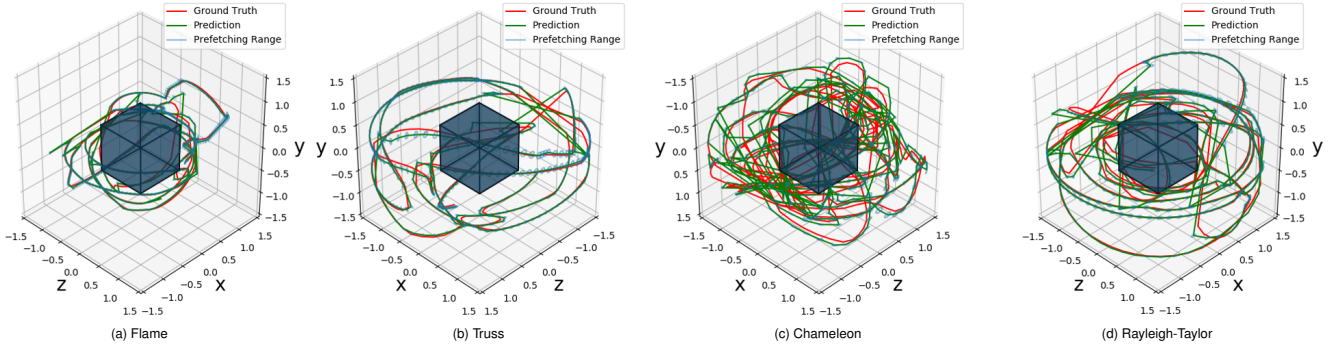

Fig. 11: Comparison between the ground truth testing trajectories and predicted trajectories using RmdnCache in Cartesian space.

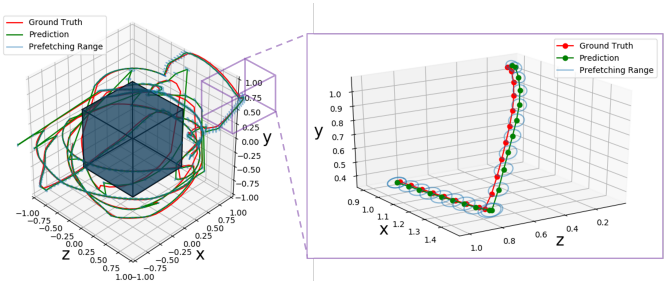

Fig. 12: Zoom-in view in Cartesian space of the testing dataset 1.

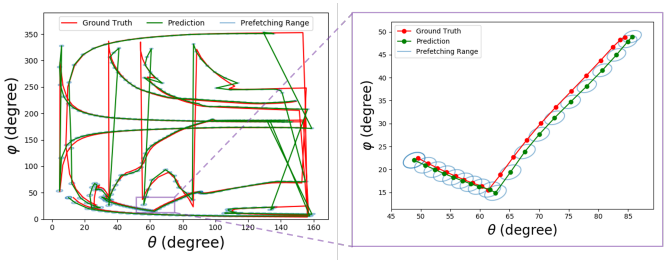

Fig. 13: Zoom-in view in Spherical space of the testing dataset 1.

starts the exploration. Once the cache reaches its maximum size (200 microblocks), existing cached microblocks will be replaced with the new incoming microblocks using LRU. The spikes are the moments where the POV dramatically changes, and the temporal locality of the cache is disrupted because all visible microblocks of such POVs have minimal or no overlap with the previous POVs. All algorithms perform poorly on those POVs.

To gain a more detailed examination of their performance, Tab. 2 shows the average miss rate (AMR) using different algorithms on all 4 testing datasets. It can be observed that all prefetching algorithms give lower miss rates than LRU where no prefetching is utilized. Our RmdnCache gives the lowest average miss rate among the prefetching algorithms for all testing datasets, showing the RmdnCache outperforms other prefetching algorithms in the accuracy of predicting visible microblocks of the next POV. In our volume rendering visualization task, the visualization configurations (transfer function, ray casting sampling distance, rendering image resolution, and so on) for all the prefetching algorithms are the same, and thus, they share the same rendering time that is only determined by the visualization configuration. Since the prefetching procedure only executes in parallel during the rendering procedure and gets stopped when the rendering is done, this means for the same POV of a particular dataset, each algorithm was assigned the same time interval to perform prefetching. The lowest miss rate of RmdnCache reveals its efficiency in prefetching more reusable microblocks for the future.

The dynamic of an exploratory trajectory determines the amount of dramatic changes in the POVs. As a result, the miss rates of all the algorithms increase from less dynamic trajectories (tests 1 and 2) to more dynamic trajectories (tests 3 and 4). For a statically sound evaluation, we also measure the 95th percentile of the miss rates of all POVs in Tab. 2. Our RmdnCache has the lowest 95th percentile meaning its prediction accuracy is not only higher but also more stable. We can also observe that deep learning methods, LSTM and RmdnCache, performs better in prefetching accuracy than a traditional predictive model, like the Markov model, and prove their potential to capture user exploration pattern better. RmdnCache performs the best because it not only predicts the next POV as LSTM does, but also has an effective range for prefetching. Although a large prefetching pool will load a broader range of microblocks for each POV with better temporal locality, it

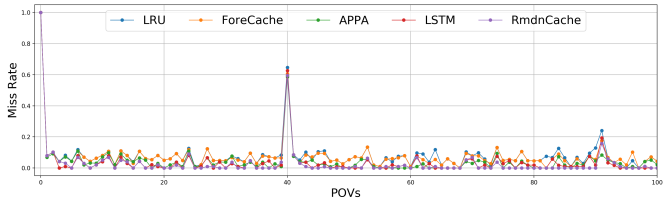

Fig. 14: Miss rate of the first 101 POVs using LRU and different prefetching algorithms on testing dataset 1.

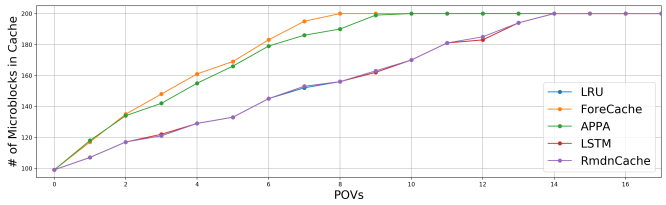

Fig. 15: Number of microblocks in cache of the first 17 POVs using LRU and different prefetching algorithms on testing dataset 1.

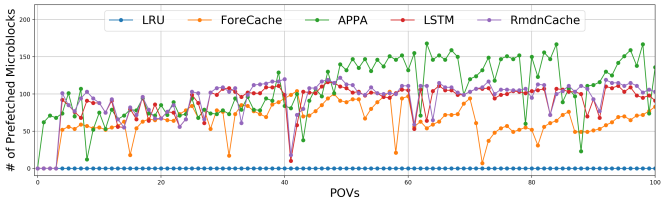

Fig. 16: Prefetched number of microblocks of the first 100 POVs using LRU and different prefetching algorithms on testing dataset 1.

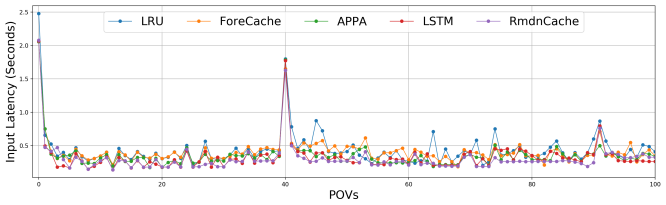

Fig. 17: Input latency of the first 101 POVs using LRU and different prefetching algorithms on testing dataset 1.

needs a longer time to load such a wide range of microblocks during prefetching. This will result in unfinished prefetching due to early stops and the limited loaded microblocks might not be the ones that will be reused for the next POV. On the other hand, a small prefetching pool will not have an unfinished prefetching issue, but it will not update the cache with enough microblocks of the next POV due to its limited pool of predicted microblocks. This is one of the main issues of APPA where its prefetching range is predefined without the knowledge of user exploration patterns. Fig. 16 shows how many microblocks get prefetched during the rendering stage using different algorithms for each POV (the first 101 POVs are shown here, and the full record can be found in the appendix). LRU is constant zero as there is no prefetching. Predictive model Forecache, LSTM, and RmdnCache will not do any prefetching until a window size of POVs is available. We can observe that APPA tries to prefetch the most microblocks compared with other algorithms. RmdnCache addresses this problem by using user data to learn a suitable range, and its anticipated prefetching range adjusts to the input, making it a more effective solution. The miss rate result on SSD is shown in Tab. 3. The miss rate performance of APPA, LSTM, and RmdnCache get improved due to the faster I/O time between storage and system memory, which results in more prefetched block before the deadline, thereby enhancing the subsequent caching hit rate. The miss rate of LRU does not benefit from the faster I/O because no prefetching is used. Miss rate performance of ForeCache does not change much due to its relatively large prediction error. RmdnCache still performs the best in miss rate using SSD.

4.6.2 Input Latency

We also measure the input latency of each algorithm on the testing datasets to evaluate the responsiveness for interactive visualization. Fig. 17 is an example plot of the input latency of the first 101 POVs on testing dataset 1 using LRU and different prefetching algorithms. The detailed input latency of all POVs for each testing dataset can be found in the appendix. As expected, we observed a similar distribution of spikes as the miss rate in Fig. 14, because the miss rate is the main factor in determining the input latency. The first POV gives the longest input latency due to the 100% miss rate. All the algorithms share the same cache copy time from RAM to VRAM. Since the upper limit of the cache is only 200 microblocks, the average memory copy time is around 0.003 seconds which is much smaller compared to other steps like caching and rendering.

Tab. 4 provides a closer comparison by showing the average input latency with different algorithms on all 4 testing datasets. Similar to the miss rate, all prefetching algorithms give shorter input latency than LRU. Our RmdnCache gives the lowest input latency among all prefetching algorithms for all testing datasets, showing RmdnCache can further improve the responsiveness of the interactive visualization system. We also measure the 95th percentile for each algorithm and find our RmdnCache gives the lowest value, meaning its improvement in the input latency is not only the highest but also consistent throughout the POV trajectories. The input latency result on SSD is shown in Tab. 5. All algorithms benefit from the faster I/O time that greatly improves the caching performance. RmdnCache still performs the best in input latency using SSD.

During the prefetching procedure, two consecutive steps are carried out: 1) Retrieving microblocks to prefetch for next POV; 2) Performing prefetching for those microblocks on Cache. The total time budget allowed for those two steps together is determined by the rendering time. When rendering of a particular POV is done, the second prefetching step has to be preempted, even if it is incomplete. In order to save time for step 2) to get more microblocks prefetched, step 1) has to be optimized to use as minimal time as possible. There are two sub-steps for step 1: i) Making a prediction for prefetching; ii) Find visible microblocks from the prediction. The pre-trained RmdnCache model is loaded as torch script and conducts inferencing in real-time through LibTorch under a C++ environment. Due to the compact size of our traced model using optimized hyperparameters, which is 199KB in total for RNN and MDN networks together with around 42,000 parameters, the first sub-step of making inferencing can be performed efficiently on CPU with a negligible overhead, which is around 0.5 *ms*. As a result, the inferencing time using our RmdnCache model only takes approximately 1.5% (3%) of the required input latency for a typical responsive, interactive visualization system with 30 (60) FPS. In addition, we make use of a lookup table to optimize the search process in sub-step 2 of the multi-resolution volume rendering framework. As a result, the latency of setp 1) is reduced from an average of 0.02 seconds to 0.003 seconds for all testing datasets running on our hardware.

A visualization of the user's exploration of the Flame dataset using multi-resolution volume rendering can be found in the appendix, where three randomly selected POVs are checked for miss rate and input latency.

4.6.3 Prefetching Time Budget

The prefetching time budget is the time allocated for prefetching, which equals the rendering time on the GPU. If the rendering quality is not under constraint, a renderer can push its frame rate by adjusting rendering time on GPU through control parameters. However, the upper bound of the frame rate is mainly determined by the caching time when dealing with large-scale datasets. The rendering time is mainly determined by three factors: the time complexity of the chosen visualization algorithm,

Table 2: Average miss rate (AMR) and its 95th percentile using LRU and four prefetching algorithms on testing datasets stored on HDD.

Testing Dataset	LRU		ForeCache		APPA		LSTM		RmdnCache	
	AMR ↓	95th P ↓	AMR ↓	95th P ↓	AMR ↓	95th P ↓	AMR ↓	95th P ↓	AMR ↓	95th P ↓
Flame	9.357%	20.173%	8.361%	19.007%	5.846%	18.842%	4.464%	14.307%	**3.566%**	**12.393%**
Truss	10.481%	26.264%	9.952%	23.937%	5.924%	20.212%	5.444%	21.624%	**4.677%**	**18.829%**
Chameleon	26.451%	54.865%	24.895%	51.322%	21.482%	51.518%	19.228%	51.622%	**18.653%**	**50.507%**
Rayleigh-Taylor	16.526%	41.143%	15.427%	38.262%	11.545%	36.337%	9.791%	35.102%	**8.917%**	**34.696%**

Table 3: Average miss rate (AMR) and its 95th percentile using LRU and four prefetching algorithms on testing datasets stored on SSD.

Testing Dataset	LRU		ForeCache		APPA		LSTM		RmdnCache	
	AMR ↓	95th P ↓	AMR ↓	95th P ↓	AMR ↓	95th P ↓	AMR ↓	95th P ↓	AMR ↓	95th P ↓
Flame	9.357%	20.173%	8.765%	20.238%	3.139%	13.295%	3.051%	8.425%	**2.193%**	**7.143%**
Truss	10.481%	26.264%	10.770%	26.244%	3.339%	15.972%	3.855%	16.289%	**2.915%**	**12.512%**
Chameleon	26.451%	54.865%	24.817%	50.514%	18.408%	51.581%	12.741%	50.027%	**11.362%**	**48.388%**
Rayleigh-Taylor	16.526%	41.143%	16.086%	40.828%	7.694%	37.753%	6.848%	37.662%	**5.657%**	**34.782%**

Table 4: Average input latency (AIL) in seconds and 95th percentile using LRU and four prefetching algorithms on testing datasets stored on HDD.

Testing Dataset	LRU		ForeCache		APPA		LSTM		RmdnCache	
	AIL ↓	95th P ↓	AIL ↓	95th P ↓	AIL ↓	95th P ↓	AIL ↓	95th P ↓	AIL ↓	95th P ↓
Flame	0.538	0.832	0.447	0.717	0.419	0.698	0.387	0.648	**0.360**	**0.577**
Truss	0.588	0.965	0.560	0.938	0.491	0.788	0.478	0.759	**0.468**	**0.745**
Chameleon	1.027	1.713	0.958	1.577	0.909	1.584	0.834	1.552	**0.833**	**1.492**
Rayleigh-Taylor	0.461	1.439	0.443	1.376	0.39	1.192	0.36	1.137	**0.354**	**1.089**

Table 5: Average input latency (AIL) in seconds and 95th percentile using LRU and four prefetching algorithms on testing datasets stored on SSD.

Testing Dataset	LRU		ForeCache		APPA		LSTM		RmdnCache	
	AIL ↓	95th P ↓	AIL ↓	95th P ↓	AIL ↓	95th P ↓	AIL ↓	95th P ↓	AIL ↓	95th P ↓
Flame	0.224	0.329	0.206	0.328	0.207	0.313	0.196	0.299	**0.184**	**0.271**
Truss	0.304	0.397	0.296	0.403	0.295	0.391	0.300	0.408	**0.274**	**0.363**
Chameleon	0.215	0.299	0.206	0.319	0.206	0.324	0.195	0.302	**0.189**	**0.277**
Rayleigh-Taylor	0.188	0.257	0.188	0.257	0.184	0.250	0.181	0.243	**0.167**	**0.240**

the resolution of the resulting image, and the capability of the GPU. In order to investigate how the prefetching algorithms perform under different prefetching time budgets, we select the sample distance on the ray of the ray-casting based direct volume rendering (DVR) as the independent parameter to control the time complexity of the visualization algorithm while keeping the other two factors the same.

Less prefetching time budget results in more insufficient prefetching as there is no enough time allowed to wait for the finish of prefetching predicted microblocks, instead, the process gets preempted by an early stop. We record such events of all prefetching algorithms by counting the occurrence of early stops among all the POVs, as shown in Fig. 18a. As the prefetching time budget decreases, an increasing number of POVs suffer from insufficient prefetching caused by early stops. For a prefetching algorithm, its early stop count is determined by the size of the predicted prefetching pool and the accuracy of the prediction. A larger prefetching pool requires more time to load its microblocks and results in a higher chance of an early stop. A more accurate prediction will encounter less misses when updating the cache during prefetching, leading to less data movement with lower chance of an early stop. APPA is most affected due to its relatively larger size of the prefetching pool for each POV through range coverage and its lower prediction accuracy. Because both ForeCache and LSTM retrieve prefetched microblocks candidates through a single-point coverage, they have smaller prefetching pools and are not affected as negatively as APPA. ForeCache experiences a higher number of misses than LSTM when updating the cache, which is due to its lower prediction accuracy compared to LSTM. This leads to a higher amount of data movement and a greater count of early stops for ForeCache. Despite RmdnCache having a larger prefetching pool than ForeCache and LSTM, thanks to its ability to cover multiple points, it stops earlier less frequently than ForeCache because it makes more precise predictions. It should be noted that a greater number of early stops does not always indicate poor prefetching performance. Nevertheless, an effective prefetching algorithm must make the most of the allocated prefetching time budget. Our RmdnCache experiences a higher number of early stops than LSTM as it prefetches more microblocks to make full use of the time budget.

Fig. 18b shows the average miss rates as prefetching time budget increases. LRU gives the same highest average miss rate for it does not have the prefetching step. Our RmdnCache always gives the lowest average miss rate compared to other prefetching algorithms. Fig. 18c shows the changes in the average caching time, which is the main contributor to the overall input latency. LRU gives the highest average caching time due to the lack of prefetching, while RmdnCache gives the lowest. Since the caching time is determined by the miss rate, the measured average caching time results are compliant with the average miss rate results in Fig. 18b. The input latency is close to the sum of the caching time and the rendering time, which is also the prefetching time budget. As shown in Fig. 18d, our RmdnCache also always gives the lowest average input latency among all prefetching algorithms. As prefetching time budget gets smaller, all prefetching algorithms will converge to the performance of the LRU on both average miss rate and input latency, which means the benefit of doing prefetching is diminished in such an extreme case. However, a typical visualization on large-scale datasets normally requires a reasonable amount of rendering time for a high-quality rendering result, our proposed prefetching algorithm can help improve the responsiveness of such a visualization system.

4.6.4 Sampling Density of cPOVs

We also investigate how the number of cPOVs, sampled on the prefetching range contour, would impact the overall performance of RmdnCache. We increase the number of cPOVs from 0, where no cPOV are sampled, to 120 and measure the average miss rate and the average input latency of RmdnCache, as shown in Fig. 19. When no cPOVs are sampled, RmdnCache shows suboptimal performance in terms of both average miss rate and average input latency. In this case, RmdnCache operates similarly to the LSTM method, where only one predicted POV is used to derive a single-point coverage of microblocks, resulting in insufficient prefetching. RmdnCache delivers better results when more sampled cPOVs are included, allowing it to retrieve more prefetching microblocks through multiple point coverages. However, as the number of cPOVs increases, RmdnCache's performance becomes suboptimal again due to the overhead of combining too many point coverages. This overhead grows proportionally with the number of cPOVs and consumes a considerable amount of precious prefetching time budget as too many cPOVs are included. This also causes insufficient prefetching due to the limited time available for prefetching operations. It is hard to select one single number of cPOVs for different datasets. And the guideline is to find the local minima of the miss rate from the U shape relationship between the average miss rate/input latency and the number of cPOVs.

5 Discussion

From the results, we observed that the trained RmdnCache model performs well on exploratory trajectories, including ones from new users on unseen volumes. We try to explain such an observation here. The shape of an exploratory trajectory is related to several factors: 1) A volumetric dataset itself will affect trajectories for exploration as different volumes will have distinct spatial features. 2) Data-dependent operations, such as transfer functions or query-based visualization, will likewise change the final visualization, leading to subsequent alterations in a user's points of interest. 3) Users with diverse backgrounds and learning preferences will also shape the manner in which they interact with a dataset. We utilize two popular 3D trajectory metrics in robotic path planing [12,39,62], length and smoothness, to measure the distance between trajectory in feature space. Please refer to the appendix for their detailed definitions. Fig. 20 shows the 4 testing trajectories and all training trajectories used to derive the training data. We can observe that all trajectories globally are distinct from each other. As a result, it is infeasible to train from and predict global trajectories directly.

However, how our model works is to predict the next POV from a brief sequence of preceding POVs locally, rather than considering the whole trajectory globally. The optimal size of such sequence is as small as 3, which is also the window size of the input sequence of the LSTM

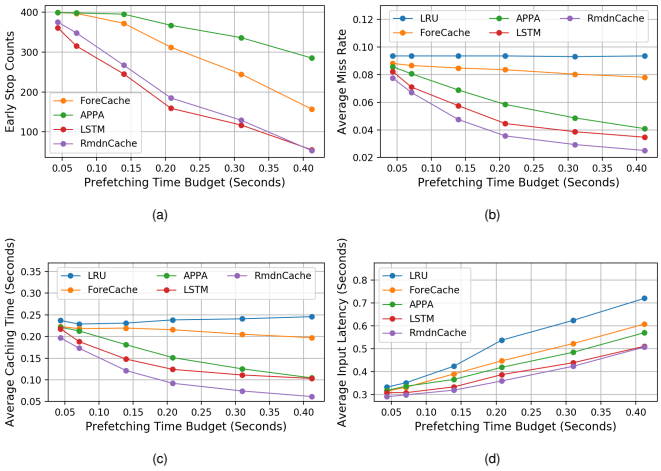

Fig. 18: Examine the differences between LRU and different prefetching algorithms regarding early stop counts (a), average miss rate (b), average caching time (c), and average input latency (d), as prefetching time budget (rendering time) increases.

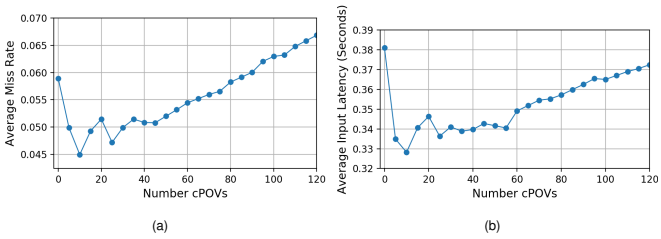

Fig. 19: Miss rate and input latency performance of RmdnCache using a different number of sampled cPOVs on testing dataset 1.

model followed by the MDN model as discussed in Sec. 4.5. As a result, our predictive model is working on a very short local segment of the entire trajectory. While different volumes, users, and data operations will inevitably influence an exploratory trajectory, the local dynamics within a short sequence of POVs still adhere to the spatial variations captured by the trained parameters of the predictive model. This is because only three view-dependent operations (rotating, zooming, and panning) are involved in generating the local sequences. Another reason is that the user's exploration time series exhibits correlation within the spatial domain without random jumps in space. For instance, the location of a given POV within a trajectory tends to be correlated with its preceding POVs in terms of both distance and direction. As a result, the position of the POV is not far from its preceding POV, and its angle aligns with the directional trend observed in the preceding POV sequence. This assumption can be proved from Fig. 21a to Fig. 21d where sequences generated from each testing trajectory are closer to each other, and they form a distribution with a maximum likelihood (distribution mode) in the feature space. Fig. 21e contains all the training sequences derived from our training trajectories together with the mode of the four testing sequence distributions. From the figure, we can observe that the testing sequences are comparable with the training sequences, so the features of testing sequences can be learned from training sequences. We can conclude that even though the training sequences come from the trajectories of different users exploring different volumetric datasets, the RmdnCache can still effectively learn the parameters to predict the sequences for new users exploring unseen volumetric datasets.

6 Conclusion

We present RmdnCache, a prefetching neural network that improves the efficiency of microblock prefetching in interactive large-scale volume visualization using a multi-resolution visualization framework. We have developed a deep learning network model that more accurately predicts

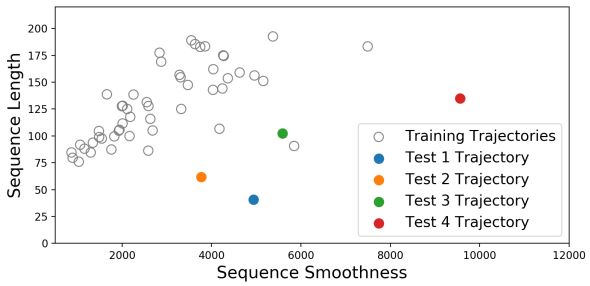

Fig. 20: Distribution of testing and training trajectories in feature space.

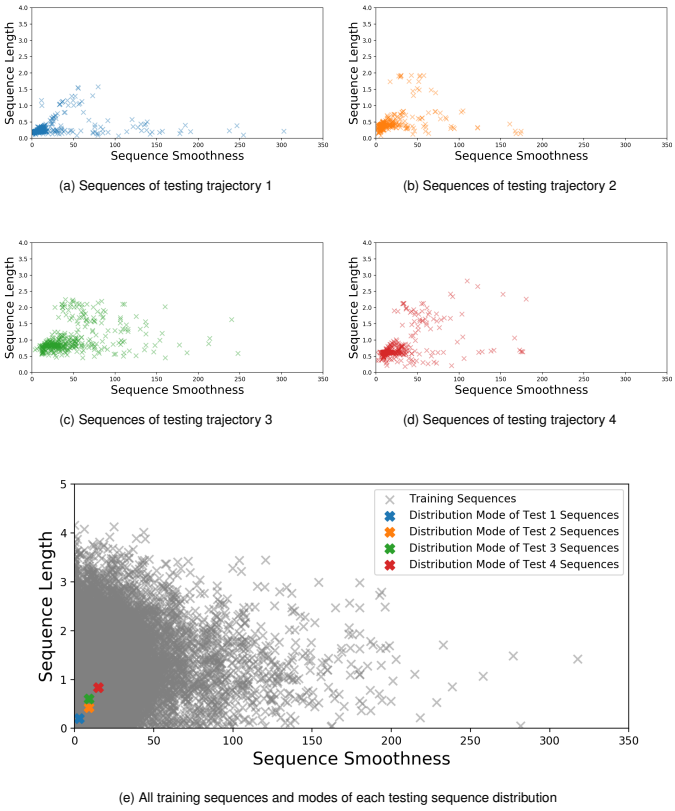

Fig. 21: Distribution of testing and training sequences in feature space.

the location of the next POV and its prefetching range by training it on representative datasets that record users' exploration patterns. We have optimized the network's key hyperparameters to enhance its performance. To demonstrate the effectiveness of our approach, we compared it to existing state-of-the-art prefetching algorithms relevant to our problem. RmdnCache achieved the lowest cache miss rate and overall input latency, resulting in a responsive and interactive visualization experience for large-scale volumetric datasets. We view this work as an effort to enhance data visualization performance on edge devices with limited computational capabilities through efficient deep learning-based prefetching. In the future, we aim to explore other seq2seq models like the bidirectional LSTM and Transformer with self-attention layers, to identify the network architecture that performs best in 3D visualization use cases. Additionally, we plan to redesign the network for end-to-end training to streamline the training and inference process. We also intend to conduct a more detailed study on the global and local behaviors of user explorations and their impacts on predictive model designs and performance.

Acknowledgement

This research has been sponsored in part by the National Science Foundation grant IIS-1652846 and Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contracts DE-AC02-06CH11357, program manager Margaret Lentz. The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] C. L. Bajaj, V. Pascucci, and D. R. Schikore. Fast isocontouring for improved interactivity. In *Proceedings of 1996 Symposium on Volume Visualization*, pp. 39–46, 1996. doi: 10.1109/SVV.1996.558041 2

[2] C. L. Bajaj, V. Pascucci, D. Thompson, and X. Y. Zhang. Parallel accelerated isocontouring for out-of-core visualization. In *Proceedings 1999 IEEE Parallel Visualization and Graphics Symposium (Cat. No.99EX381)*, pp. 97–122, 1999. doi: 10.1109/PVGS.1999.810144 2

[3] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. Tthresh: Tensor compression for multidimensional visual data. *IEEE transactions on visualization and computer graphics*, 26(9):2891–2903, 2019. 1

[4] M. Balsa Rodríguez, E. Gobbetti, J. Iglesias Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. Suter. State-of-the-art in compressed gpu-based direct volume rendering. *Computer Graphics Forum*, 33(6):77–100, 2014. doi: 10.1111/cgf.12280 2

[5] L. Battle, R. Chang, and M. Stonebraker. Dynamic prefetching of data tiles for interactive visualization. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, p. 1363–1375. Association for Computing Machinery, New York, NY, USA, 2016. doi: 10.1145/2882903.2882919 1, 2, 6

[6] M. Berger, J. Li, and J. A. Levine. A generative model for volume rendering. *IEEE transactions on visualization and computer graphics*, 25(4):1636–1650, 2018. 2

[7] J. Beyer, M. Hadwiger, and H. Pfister. State-of-the-art in gpu-based large-scale volume visualization. *Computer Graphics Forum*, 34(8):13–37, 2015. doi: 10.1111/cgf.12605 2

[8] C. M. Bishop. Mixture density networks. Technical report, 1994. 4

[9] U. D. Bordoloi and H.-W. Shen. View selection for volume rendering. In *VIS 05. IEEE Visualization, 2005.*, pp. 487–494. IEEE, 2005. 6

[10] P. Braun and H. Litz. Understanding memory access patterns for prefetching. In *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA*, 2019. 1

[11] R. Bruckschen, F. Kuester, B. Hamann, and K. I. Joy. Real-time out-of-core visualization of particle traces. In *Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, PVG '01, p. 45–50. IEEE Press, 2001. 2

[12] Y. Chen, J. Wu, C. He, and S. Zhang. Intelligent warehouse robot path planning based on improved ant colony algorithm. *IEEE Access*, 11:12360–12367, 2023. doi: 10.1109/ACCESS.2023.3241960 10

[13] Y. . Chiang, C. T. Silva, and W. J. Schroeder. Interactive out-of-core isosurface extraction. In *Proceedings Visualization '98 (Cat. No.98CB36276)*, pp. 167–174, 1998. doi: 10.1109/VISUAL.1998.745299 2

[14] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models. *ACM Trans. Graph.*, 23(3):796–803, aug 2004. doi: 10.1145/1015706.1015802 2

[15] M. Cox, N. Bhandari, and M. Shantz. Multi-level texture caching for 3d graphics hardware. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, ISCA '98, p. 86–97. IEEE Computer Society, USA, 1998. doi: 10.1145/279358.279372 1

[16] M. Cox and D. Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proceedings of the 8th Conference on Visualization '97*, VIS '97, p. 235–ff. IEEE Computer Society Press, Washington, DC, USA, 1997. 2

[17] P. R. Doshi, Geraldine E, G. E. Rosario, E. A. Rundensteiner, and M. A. Ward. A strategy selection framework for adaptive prefetching in data visualization. In *15th International Conference on Scientific and Statistical Database Management, 2003.*, pp. 107–116, 2003. doi: 10.1109/SSDM.2003.1214972 1

[18] P. R. Doshi, E. A. Rundensteiner, and M. O. Ward. Prefetching for visual data exploration. In *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications*, DASFAA '03, p. 195. IEEE Computer Society, USA, 2003. 1

[19] R. Farias, J. S. B. Mitchell, and C. T. Silva. Zsweep: An efficient and exact projection algorithm for unstructured volume rendering. In *Proceedings of the 2000 IEEE Symposium on Volume Visualization*, VVS '00, p. 91–99. Association for Computing Machinery, New York, NY, USA, 2000. doi: 10.1145/353888.353905 2

[20] R. Farias and C. T. Silva. Out-of-core rendering of large, unstructured grids. *IEEE Computer Graphics and Applications*, 21(4):42–50, 2001. doi: 10.1109/38.933523 2

[21] J. Gao, J. Huang, C. Johnson, and S. Atchley. Distributed data management for large volume visualization. In *VIS 05. IEEE Visualization, 2005.*, pp. 183–189, 2005. doi: 10.1109/VISUAL.2005.1532794 2

[22] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14346–14355, 2021. 2

[23] E. Gobbetti and F. Marton. Far voxels: A multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, p. 878–885. Association for Computing Machinery, New York, NY, USA, 2005. doi: 10.1145/1186822.1073277 1

[24] S. Guthe and W. Strasser. Advanced techniques for high-quality multiresolution volume rendering. *Computers & Graphics*, 28(1):51–58, 2004. doi: 10.1016/j.cag.2003.10.018 1, 3

[25] M. Hashemi, K. Swersky, J. Smith, G. Ayers, H. Litz, J. Chang, C. Kozyrakis, and P. Ranganathan. Learning memory access patterns. In J. Dy and A. Krause, eds., *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, pp. 1919–1928. PMLR, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. 1

[26] P. Heckbert and M. Garland. Multiresolution modeling for fast rendering. In *Graphics Interface*, pp. 43–43. Canadian Information Processing Society, 1994. 1

[27] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735 3, 6

[28] F. Hong, J. Zhang, and X. Yuan. Access pattern learning with long short-term memory for parallel particle tracing. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 76–85, 2018. doi: 10.1109/PacificVis.2018.00018 1, 2, 6

[29] C. Ioannidis and A.-M. Boutsi. Multithreaded rendering for cross-platform 3d visualization based on vulkan api. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 57–62, 2020. 1

[30] B. Jeong, P. A. Navrátil, K. P. Gaither, G. Abram, and G. P. Johnson. Configurable data prefetching scheme for interactive visualization of large-scale volume data. In P. C. Wong, D. L. Kao, M. C. Hao, C. Chen, R. Kosara, M. A. Livingston, J. Park, and I. Roberts, eds., *Visualization and Data Analysis 2012*, vol. 8294, pp. 204 – 217. International Society for Optics and Photonics, SPIE, 2012. doi: 10.1117/12.910926 1

[31] C. Köse and A. Chalmers. Memory management strategies for parallel volume rendering. In *Proceedings of the 19th World Occam and Transputer User Group Technical Meeting on Parallel Processing Developments*, WoTUG '96, p. 113–125. IOS Press, NLD, 1996. 2

[32] S. Leutenegger and K.-L. Ma. Fast retrieval of disk-resident unstructured volume data for visualization. *External Memory Algorithms and Visualization*, 50, 1999. 2

[33] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, A. M. Gok, J. Tian, J. Deng, J. C. Calhoun, D. Tao, et al. Sz3: A modular framework for composing prediction-based error-bounded lossy compressors. *IEEE Transactions on Big Data*, 9(2):485–498, 2022. 1

[34] Z. C. Lipton. A critical review of recurrent neural networks for sequence learning. *ArXiv*, abs/1506.00019, 2015. 3

[35] G. Lochmann, B. Reinert, A. Buchacher, and T. Ritschel. Real-time novel-view synthesis for volume rendering using a piecewise-analytic representation. In *VMV'16 Proceedings of the Conference on Vision, Modeling and Visualization*, vol. 2016. Eurographics, 2016. 2

[36] J. A. Lorenzon and E. W. G. Clua. A novel multithreaded rendering system based on a deferred approach. In *2009 VIII Brazilian Symposium on Games and Digital Entertainment*, pp. 168–174, 2009. doi: 10.1109/SBGAMES.2009.27 1

[37] Y. Lu, K. Jiang, J. A. Levine, and M. Berger. Compressive neural representations of volumetric scalar fields. In *Computer Graphics Forum*, vol. 40, pp. 135–146. Wiley Online Library, 2021. 2

[38] F. Marton, M. Agus, and E. Gobbetti. A framework for gpu-accelerated exploration of massive time-varying rectilinear scalar volumes. *Computer Graphics Forum*, 38(3):53–66, 2019. doi: 10.1111/cgf.13671 2

[39] C. Miao, G. Chen, C. Yan, and Y. Wu. Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm. *Computers & Industrial Engineering*, 156:107230, 2021. doi: 10.1016/j.cie.2021.107230 10

[40] S. Mittal. A survey of recent prefetching techniques for processor caches. *ACM Comput. Surv.*, 49(2), Aug. 2016. doi: 10.1145/2907071 1

[41] H. Mohammed, Z. Wei, E. Wu, and R. Netravali. Continuous prefetch for interactive data applications. *Proc. VLDB Endow.*, 13(12):2297–2311, July 2020. doi: 10.14778/3407790.3407826 1

[42] V. Pascucci and R. J. Frank. Global static indexing for real-time exploration of very large regular grids. In *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, SC '01, p. 2. Association for Computing Machinery, New York, NY, USA, 2001. doi: 10.1145/582034.582036 2

[43] A. Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999. 6

[44] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics*, 1993. 3

[45] L. Shen, J. Pauly, and L. Xing. Nerp: implicit neural representation learning with prior embedding for sparsely sampled image reconstruction. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 2

[46] Shyh-Kuang Ueng, C. Sikorski, and Kwan-Liu Ma. Out-of-core streamline visualization on large unstructured meshes. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):370–380, 1997. doi: 10.1109/2945.646239 2

[47] R. Sicat, J. Krüger, T. Möller, and M. Hadwiger. Sparse pdf volumes for consistent multi-resolution volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2417–2426, 2014. doi: 10.1109/TVCG.2014.2346324 2

[48] C. Silva, Y.-J. Chiang, J. El-Sana, and P. Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. In *IEEE Visualization Course Notes*, 11 2002. 2

[49] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473, 2020. 2

[50] P. D. Sulatycke and K. Ghose. A fast multithreaded out-of-core visualization technique. In *Proceedings 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing. IPPS/SPDP 1999*, pp. 569–575, 1999. doi: 10.1109/IPPS.1999.760534 2

[51] J. Sun, D. Lenz, H. Yu, and T. Peterka. Scalable volume visualization for big scientific data modeled by functional approximation. In *2023 IEEE International Conference on Big Data (BigData)*, pp. 905–914, 2023. doi: 10.1109/BigData59044.2023.10386434 1

[52] J. Sun, D. Lenz, H. Yu, and T. Peterka. Mfa-dvr: direct volume rendering of mfa models. *Journal of Visualization*, 27(1):109–126, 2024. 1

[53] S. Suter, M. Makhynia, and R. Pajarola. Tamresh – tensor approximation multiresolution hierarchy for interactive volume visualization. *Computer Graphics Forum*, 32(3pt2):151–160, 2013. doi: 10.1111/cgf.12102 2

[54] S. K. Suter, J. A. Iglesias Guitian, F. Marton, M. Agus, A. Elsener, C. P. Zollikofer, M. Gopi, E. Gobbetti, and R. Pajarola. Interactive multiscale tensor reconstruction for multiresolution volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2135–2143, 2011. doi: 10.1109/TVCG.2011.214 2

[55] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, p. 3104–3112. MIT Press, Cambridge, MA, USA, 2014. 3

[56] P. M. Sutton and C. D. Hansen. Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):98–107, 2000. doi: 10.1109/2945.856992 2

[57] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11358–11367, 2021. 2

[58] D. Tang, S. Singh, P. A. Chou, C. Hane, M. Dou, S. Fanello, J. Taylor, P. Davidson, O. G. Guleryuz, Y. Zhang, et al. Deep implicit volume compression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1293–1303, 2020. 2

[59] P.-P. Vázquez, E. Monclús, and I. Navazo. Representative views and paths for volume models. In *International symposium on smart graphics*, pp. 106–117. Springer, 2008. 6

[60] C. Wang and J. Han. Dl4scivis: A state-of-the-art survey on deep learning for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2022. doi: 10.1109/TVCG.2022.3167896 2

[61] M. Weiler, R. Westermann, C. Hansen, K. Zimmermann, and T. Ertl. Level-of-detail volume rendering via 3d textures. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pp. 7–13, 2000. 2

[62] Y. Xue and J.-Q. Sun. Solving the path planning problem in mobile robotics with the multi-objective evolutionary algorithm. *Applied Sciences*, 8(9), 2018. doi: 10.3390/app8091425 10

[63] Yi-Jen Chiang and C. T. Silva. I/O optimal isosurface extraction. In *Proceedings. Visualization '97 (Cat. No. 97CB36155)*, pp. 293–300, 1997. doi: 10.1109/VISUAL.1997.663895 2

[64] L. Yu, Hongfeng Yu, Hong Jiang, and Jun Wang. An application-aware data replacement policy for interactive large-scale scientific visualization. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1216–1225, 2017. doi: 10.1109/IPDPSW.2017.16 1, 2, 6

[65] Y. Yu, X. Si, C. Hu, and J. Zhang. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation*, 31(7):1235–1270, 07 2019. doi: 10.1162/neco_a_01199 4

Jianxin Sun is a Ph.D. candidate in the School of Computing at the University of Nebraska-Lincoln. He received an MS degree in Electrical and Computer Engineering from Purdue University and a BS degree in Electrical Engineering from Harbin Institute of Technology. His research interests are computer vision, machine learning, data modeling, and visualization. His current research focuses on visualizing large-scale scientific multi-dimensional datasets by leveraging deep learning techniques.

Xinyan Xie is a Ph.D. candidate in the School of Computing at the University of Nebraska-Lincoln. She received an MS degree in Statistics from the same university. Her research includes machine learning, computational geometry, 3D object reconstruction, and detection. Her current research focuses on feature learning from hyperspectral data, detection, localization, and classification of 3D objects in the form of point clouds, and machine learning applications in plant science.

Hongfeng Yu is an associate professor in the School of Computing at the University of Nebraska-Lincoln. He received a Ph.D. degree in Computer Science from the University of California-Davis. Dr. Yu's research concentrates on big data analysis and visualization, high-performance computing, and user interfaces and interaction.