FlowHON: Representing Flow Fields Using Higher-Order Networks

Nan Chen, Zhihong Li, and Jun Tao*, *Member, IEEE*

Abstract—Flow fields are often partitioned into data blocks for massively parallel computation and analysis based on blockwise relationships. However, most of the previous techniques only consider the first-order dependencies among blocks, which is insufficient in describing complex flow patterns. In this work, we present FlowHON, an approach to construct higher-order networks (HONs) from flow fields. FlowHON captures the inherent higher-order dependencies in flow fields as nodes and estimates the transitions among them as edges. We formulate the HON construction as an optimization problem with three linear transformations. The first two layers correspond to the node generation and the third one corresponds to edge estimation. Our formulation allows the node generation and edge estimation to be solved in a unified framework. With FlowHON, the rich set of traditional graph algorithms can be applied without any modification to analyze flow fields, while leveraging the higher-order information to understand the inherent structure and manage flow data for efficiency. We demonstrate the effectiveness of FlowHON using a series of downstream tasks, including estimating the density of particles during tracing, partitioning flow fields for data management, and understanding flow fields using the node-link diagram representation of networks.

Index Terms—Flow visualization, higher-order network, data transformation, data partition, and task distribution.

1 Introduction

Flow visualization plays a vital role in understanding dynamic systems for various domains and applications. In the past decades, flow visualization has been studied extensively, and many techniques were developed to effectively visualize and analyze flow fields. Recently, due to the increasing size and complexity of simulated flow fields, many approaches partition the flow data into data blocks for further processing or analysis. The data partitioning reduces the size of data processed by each computing node for scalability and allows the structure of flow fields to be understood at the block level. At the core of these techniques, a graph is used either as a data structure for graph algorithms to analyze flow fields or as a visual representation to enable clear observation and easy interaction in 2D.

Although developed for different scenarios, existing graph-based techniques usually share a similar construction process. Nodes in a graph represent data blocks, and edges represent the transition probabilities among data blocks. The transition probabilities are estimated empirically based on the number of particles moving between blocks. In this way, the graph provides affinity relationships between blocks and captures the structure of the flow field. Analysis of the graph facilitates a series of downstream tasks, including data partition [26], data prefetching [11], [13], [43], and particle advection scheduling [4]. By applying layout algorithms, the graph may be used to present the flow structure compactly without occlusion [39] as well.

However, conventional graph-based flow visualization techniques usually assume the Markovian process in describing the relationships among blocks, which could be inaccurate. The Markovian assumption implies that the particles in the same block will follow the same transition probability distribution when moving to the next block. This assumption does not hold in most cases, especially for blocks with complex flow behaviors. Some approaches may employ a multi-resolution partitioning strategy to further divide the complicated block. For example, the FlowGraph [21] evaluates the entropy of flow directions in blocks to guide the partitioning. But this strategy may require a great amount of blocks to precisely describe the curvy boundaries of regions.

To the best of our knowledge, Zhang et al. [43] is the only existing approach that considers higher-order dependencies. But this technique fails to incorporate the connections among higher-order dependencies. It captures only the local higher-order patterns but not the global structure of entire flow fields. Therefore, this technique may not be easily extended to support tasks such as data partitioning, workload balancing, and decomposition of flow fields, where the global structure matters.

In this paper, we aim to capture block-wise higher-order dependencies in a flow field at a global scale. Toward this end, we extract the higher-order dependencies among blocks and organize these dependencies as a higher-order network. The higher-order dependencies allow different flow behaviors in a single data block to be separated so that the flow transition patterns among data blocks can be accurately described. Furthermore, the network, connecting the higher-order dependencies, allows the dependencies to be studied at a larger scale and provides a compatible interface for existing network analytic algorithms to be applied. The

- * *denote the corresponding author.*
- *N. Chen is with the School of Computer Science and Engineering, Sun Yat-sen University and the Johns Hopkins University. E-mail: chenn53@mail2.sysu.edu.cn. The majority of this work was conducted during his undergraduate studies at SYSU.*
- *Z. Li and J. Tao are with the School of Computer Science and Engineering, Sun Yat-sen University and the National Supercomputer Center in Guangzhou, China. J. Tao is the corresponding author. E-mail: lizhh236@mail2.sysu.edu.cn, taoj23@mail.sysu.edu.cn.*

main challenges to achieving this goal can be summarized into three aspects. The first challenge is to extract higher-order dependencies that can precisely model diverse flow patterns in the flow field and avoid redundant higher-order dependencies at the same time. The second challenge is to approximate the transition probabilities between nodes in the network so that the flow behavior can be accurately described. The final challenge is to establish a connection between network analytic methods and flow fields, which makes it possible to analyze flow fields by analyzing corresponding higher-order networks.

To tackle the above challenges, we propose FlowHON, a unified framework to construct higher-order networks from flow fields. The framework formulates the HON construction problem as an optimization problem with three linear transformations, including two linear layers for node generation and one for transition estimation. This formulation generalizes existing HON construction algorithms. Therefore, it may lead to potentially better performance, with existing approaches being special solutions to our optimization problem. We propose an efficient approach to optimize the node generation and transition estimation in a unified framework and examine the performance of our approach using several downstream tasks with a variety of data sets. The tasks include estimating particle transitions using random walks on our network, data partitioning by applying a community detection algorithm, and visualizing the flow field structure by leveraging graph layout. We demonstrate the effectiveness by comparing our approach with existing graph-based approaches on these tasks.

2 Related Work

Graph-based techniques for flow visualization. Graph-based approaches have received considerable attention from the scientific visualization community in various kinds of applications [37]. In flow visualization, several graph-based techniques were developed to describe the access pattern among blocks during particle tracing. Bhatia et al. [2] designed edge maps for triangular meshes, which mapped the entry and exit points of streamlines on the boundary of individual triangles. Chen et al. [5] proposed the access dependency graph to assess the dependencies between different data blocks in the flow field and used it to guide the file layout for improved I/O performance. Chen et al. [3] proposed the N-hop access dependency graph that further considered the N-hop transitions. Chen et al. [4] applied discrete-time Markov chains on node-link graphs to predict particle trajectories on time-varying flow fields to guide seed advection schedules. Nouanesengsy et al. [26] utilized a flow graph with initial seed locations to estimate each data block's workload during parallel streamline generation. Guo et al. [13] developed a graph-based model that could be constructed on the fly to predict data access for data block prefetching. Gerndt et al. [11] applied a similar strategy to build a first-order probability graph that characterized the successor relation of blocks in CFD data sets. Zhang et al. [43] applied higher-order dependencies among data blocks to predict the data access pattern and guide the data prefetching. Zhang et al. [44] built an access dependency graph to estimate workload. Other works applied graphs to understand the structures of flow fields, such as Morse Connection Graph (MCG) [6], [7], Flow Web [39], FlowGraph [21], [22], Flow topology graph (FTG) [1], and Semantic Flow Graph [35].

This work, while sharing a similar idea to MCG [6], [7] of encoding flow patterns into a network's nodes, diverges in its approach and application. MCG directly analyzes the vector field of flow fields and denotes flow behaviors as nodes in a graph, with each node usually representing an irregular area. This method provides a rigorous interpretation of the underlying vector field, making it exceptionally suitable for precise visual analysis that allows the identification of subtle flow patterns. However, its computational complexity restricts its use primarily to 2D flow fields, and extending MCG to accommodate 3D or unsteady flow fields is non-trivial. In contrast, FlowHON leverages higher-order dependencies among uniformly partitioned data blocks to depict flow dynamics, employing higher-order nodes to extract flow patterns. FlowHON provides a statistical way to identify flow patterns as it is built from individual trajectories sampled from vector fields. Essentially, it relies on the statistics of particle movements to extract higher-order Markov dependencies among blocks, which improves its scalability and makes it practically applicable to 3D and unsteady flow fields. Hence, while MCG is ideal for precise visual flow analysis, our method prioritizes an approximation of the flow field to emphasize dominant behaviors and interdependencies among data blocks, which aids visual exploration and parallel particle tracing.

Parallel tracing and data management. Parallel particle tracing algorithms generally fall into three main categories: data-parallelism, task-parallelism, and hybrid-parallelism [45]. The data-parallel mechanism distributes data blocks to computing nodes and exchanges particles during tracing. Yu et al. [40] partitioned flow data based on the hierarchical representation of data blocks. Moloney et al. [23] applied k-d tree to divide the dataset of uniform grid for load balancing in sort-first parallel direct volume rendering. With a similar idea, Zhang et al. [42] applied k-d tree decomposition to balance workload during parallel particle tracing. Chen et al. [8] partitioned flow data based on flow direction and features. Peterka et al. [28] employed a static round-robin partition algorithm to distribute data blocks among processes. Nouanesengsy et al. [25] partitioned flow data into mutually exclusive spans of time for high-resolution FTLE computation. Graph-based models are generally employed to estimate the workload of each data block during running, which cooperates with workload-aware allocation methods to generate the optimal data distribution among processors [26].

The task-parallel mechanism distributes the tracing tasks to computing nodes, which load data blocks on demand. Task-parallel tracing frameworks usually integrate methods such as data prefetching and file layout rearrangement to exploit data locality and to boost I/O performance. Many of these approaches leveraged graph-based representation to guide the file layout [3], [5], data prefetching [11], [13], [43], and task grouping [4]. Hong et al. [17] employed an LSTM-based model to estimate the access pattern for parallel particle tracing in flow fields.

Particle density estimation. Reich et al. [29] applied

time-discrete Markov chains on static unstructured flow fields to estimate particle distributions over time given initial particle distributions. Hollt et al. [15] used first-order forward tracking to estimate the trajectory of a particle originating in a specific cell. Guo et al. [12] introduced a divide-and-conquer mechanism to compute stochastic flow maps, where they decoupled the time domain into short periods, performed Monte Carlo particle tracing for each subinterval independently, and then composed the results to approximate the particle distribution for a longer period.

Our approach falls into the category of graph-based approaches. However, unlike existing methods, ours is the only one that leverages higher-order dependencies and their connections to depict flow fields at a refined level. Most similar to ours are methods that consider N-hop or higher-order dependencies. Chen et al. [3] included N-hop transitions among blocks into dependency graphs. But this construction only compensates for underestimated long-term dependencies and does not provide a refined level of behaviors inside each data block. Zhang et al. [43] explored higher-order dependencies to distinguish flow behaviors in individual blocks. But these dependencies are only used for data prefetching, and their connections are not considered. Hence, this work does not describe the higher-order dependencies at a global level.

Clustering techniques for flow visualization. Clustering algorithms play a crucial role in flow visualization by identifying and summarizing representative flow patterns within a flow field. Yu et al. [41] clustered streamlines based on spatial proximity and geometric similarity, constructing a hierarchy of streamline bundles that capture flow structures at multiple levels of detail. Tao et al. [34] identified representative streamlines based on their contribution to sampled viewpoints and then performed clustering by minimizing a mutual information loss. Lu et al. [20] represented streamlines using statistical distributions of user-defined measurements along their trajectories, leveraging these extracted features for similarity-based clustering. Oeltze et al. [27] evaluated spectral clustering, agglomerative hierarchical clustering, and k-means quantitatively and qualitatively for reducing visual clutter in simulated blood flow visualization. Hong et al. [16] introduced Flow LDA, which represents pathlines as documents and features as words, applying Latent Dirichlet Allocation (LDA) to cluster pathlines through probabilistic topic modeling. Han et al. [14] proposed FlowNet, which encodes streamlines as binary volumes, uses an autoencoder to extract latent representations, and applies t-SNE followed by DBSCAN for clustering. In contrast to these approaches, FlowHON encodes flow fields into higher-order networks, where individual nodes represent small, localized flow patterns. By applying community detection to the constructed network, FlowHON identifies and groups these local patterns into more representative streamline clusters for visualization.

3 Higher-order Network for Flow

We introduce *FlowHON*, a higher-order network (HON) for flow visualization. Our goal is to encode the higher-order Markov dependencies in the graph representation, which can be leveraged by existing graph-based visualization techniques to improve their performance without modification. The higher-order dependency indicates that the transition probability relies on not only the current state but also a series of previous states. Higher-order dependency commonly exists in many real-world applications, but traditional graph-based approaches do not exploit it. Conventional approaches are often built upon first-order networks (FON), which cannot describe complicated transition patterns. In this section, we will briefly introduce the concept of HON in the context of flow visualization and explain how the HON facilitates the analysis of flow fields. For HON techniques in a broader context, please kindly refer to Appendix A.

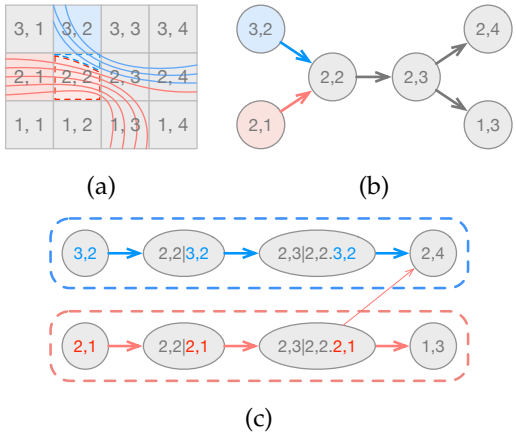


Fig. 1: Illustrative example of the higher-order network. (a) shows a vector field uniformly partitioned into a 4×3 grid. (b) shows the corresponding first-order network. (c) shows the corresponding higher-order network.

Higher-order dependency. Figure 1 (a) illustrates an example of a flow field uniformly partitioned into a 4×3 grid. The streamlines exhibit two movement patterns, which are distinguished by their colors. The trajectories of particles starting from block $(3, 2)$ are colored in blue, and those from block $(2, 1)$ are in red.

Traditional approaches (e.g., [21], [22], [39]) model the block-wise relationships by collecting statistics of sampled particles moving between consecutive blocks, resulting in a directed graph, as shown in Figure 1 (b). This graph encodes the *first-order Markov dependency*, meaning that the distribution of the next block to visit only depends on the current block where a particle resides. For example, all particles in block $(2, 2)$ will move to $(2, 3)$ (i.e, $p((2, 2) \rightarrow (2, 3)) = 1.0$), and a particle in $(2, 3)$ has an equal chance to visit either $(2, 4)$ or $(1, 3)$ (i.e., $p((2, 3) \rightarrow (2, 4)) = 0.5$ and $p((2, 3) \rightarrow (1, 3)) = 0.5$). However, this assumes that all particles in a block share the same distribution, which may be inaccurate. In Figure 1 (a), we can see that the blue particles in $(2, 3)$ will move to $(2, 4)$ and most of the red ones will move to $(1, 3)$.

The *higher-order Markov dependency* encodes the transition from a sequence of possible events to the next event. In our scenario, this means that the probability of the next block to visit depends on not only the current block where a particle resides but also the series of blocks it has visited

before. For example, using higher-order dependencies, the red particles in $(2,3)$ will be denoted as $(2,3)|(2,2).(2,1)$, meaning particles currently in $(2,3)$ given that they come from $(2,2)$ and $(2,1)$. For these particles, the probability to visit $(2,4)$ becomes $p((2,3)|(2,2).(2,1) \rightarrow (2,4)) = 0.2$ and the probability to visit $(1,3)$ becomes $p((2,3)|(2,2).(2,1) \rightarrow (1,3)) = 0.8$. Similarly, for blue particles in $(2,3)$, the transition becomes $p((2,3)|(2,2).(3,2) \rightarrow (2,4)) = 1.0$. Note that patterns of the blue and red streamlines become more distinguishable using this representation. Therefore, *the higher-order Markov dependencies provide a clearer picture of particle movements between blocks*. To avoid confusion, we refer to the evidence sequence of events as a *higher-order state* (e.g., $(2,3)|(2,2).(3,2) \rightarrow (2,4)$ is a third-order dependency and $(2,3)|(2,2).(3,2)$ is a third-order state).

Higher-order network. The higher-order dependencies only describe the local transition patterns. To further capture the global structure of a flow field, transitions among higher-order states must be incorporated. The higher-order network (HON) is a directed graph whose nodes are higher-order states and whose edges encode transition probabilities between nodes. Figure 1 (c) illustrates such an example. For particles in block $(3,2)$, as no preceding block is given, these particles start from a first-order state $(3,2)$. After moving to $(2,2)$, the particles have a second-order state $(2,2)|(3,2)$. An edge is added to the graph to connect the two states $(3,2)$ and $(2,2)|(3,2)$. Compared to the corresponding FON (Figure 1 (b)), the higher-order network splits the first-order state $(2,2)$ into two second-order states $(2,2)|(3,2)$ and $(2,2)|(2,1)$. These second-order states "record" the history of particles to better distinguish different flow behaviors. Similarly, the node $(2,3)$ is split into two nodes $(2,3)|(2,2).(3,2)$ and $(2,3)|(2,2).(2,1)$, and the edge $(2,2) \rightarrow (2,3)$ is split into two respective edges.

Why do we need higher-order networks in flow analysis? An interpretation of the HON is that *the HON implicitly subdivides blocks in a regular grid along the flow*. As illustrated by Figure 1 (a), particles in block $(2,2)$ form two groups based on which blocks they have previously visited. This implicitly subdivides block $(2,2)$ into a red and a blue region along streamlines, corresponding to flows going to the right and those going downward, respectively. Similar subdivisions of blocks are observed in real-world settings, as discussed in Section 4.2.

The subdivision behavior of the HON provides a finer-level description of the flow field, where the global structure can be better studied. In Figure 1 (c), we can easily identify two communities in the HON, where there is only a weak transition between the two communities. However, in Figure 1 (b), this structure is not available in the FON, as the two nodes $(2,2)$ and $(2,3)$ mix different movement patterns. Although multi-resolution techniques, such as octree, may be used to further subdivide blocks with complicated patterns, these techniques partition the blocks regularly along the axes, which requires a much higher number of small blocks to approximate the irregular flow boundaries.

Additionally, *as a directed graph, the HON allows all existing graph analysis approaches to be applied directly, with better accuracy*. For example, the random walk can be used to approximate the particle movement on the graph. In Figure 1, a particle starting from $(3,2)$ will reach either $(2,4)$ or $(1,3)$ with 50% of chance using the FON, while this particle will reach $(2,4)$ for sure using the HON. For another example, community detection algorithms can be used to identify the two different streamline bundles using the HON but not using the FON.

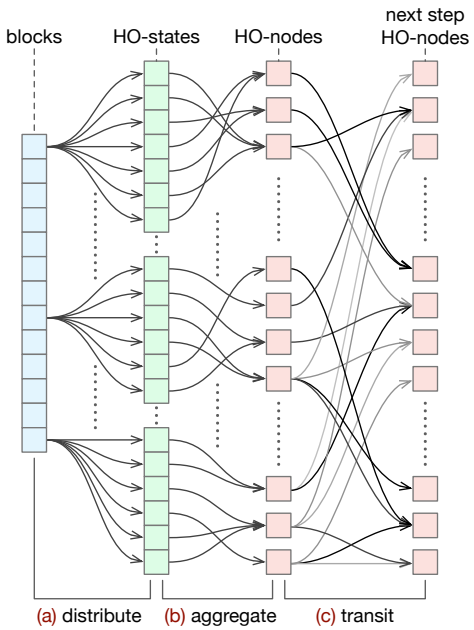


Fig. 2: Illustration of our formulation of the HON construction from data blocks, which is considered as three linear layers: (a) distributing particles from data blocks to HO-states; (b) aggregating HO-states into HO-nodes; and (c) transiting from current HO-nodes to next-step HO-nodes.

4 Our Construction Algorithm

We formulate the HON construction process as a network optimization problem. The network contains three linear layers connecting particle distributions in data blocks, higher-order states (HO-states), and higher-order nodes (HO-nodes). The HO-nodes are formed by aggregating similar HO-states to reduce the size of a fixed-order network [31]. Specifically, these three layers represent *the distribution of particles from data blocks to HO-states*, *the aggregation of HO-states into HO-nodes*, and *the transitions between HO-nodes*, respectively, as shown in Figure 2. By connecting the three layers, our approach allows different processing steps in HON construction to be optimized in a unified framework. In this section, we will introduce our problem formulation and then discuss the basic components of the optimization in detail.

Notations. For clarity, we distinguish two similar concepts as follows. The *higher-order state* is the finest level of elements in the HON. An HO-state represents a sequence of blocks visited by a particle consecutively (e.g., $(2,3)|(2,2).(3.2)$). The *higher-order node* is a node in the HON, representing a group of HO-states exhibiting similar transition behaviors. Aggregating HO-states into HO-nodes could reduce the size of the HON, enhancing computation efficiency and visualization. In the following, we will denote matrices using bold uppercase letters (e.g., $\mathbf{D}$) and vectors

using bold lowercase letters (e.g., $\mathbf{b}$). Subscripts, such as $\mathbf{D}_{i,j}$ or $\mathbf{b}_i$, will be used to refer to their elements.

Existing HON construction approaches Higher-order networks fall into two types: fixed-order [31] and variable-order networks [38]. In a fixed-order network, each node represents a state of a fixed order, and each edge denotes the transition between two states. Although capturing the finest-grained dependency information up to a fixed order, a fixed-order network usually expands exponentially with its order. Compared to fixed-order networks, a variable-order network contains nodes representing states of variable orders and adds higher-order nodes only when their transition behaviors differ from corresponding lower-order ones, reducing network size by eliminating unnecessary higher-order nodes. However, variable-order networks reduce the size based on hand-crafted rules, which may not always yield the optimal performance.

4.1 Problem formulation

We jointly consider two essential problems in the HON construction: how to generate nodes in the HON and how to estimate the transition probabilities between nodes. Toward this end, we formulate the HON construction process as an optimization problem for a three-layer network. Each layer is a linear transformation represented by a weight matrix, which is elaborated as follows.

Distribution. The first layer is the distribution layer, represented by the *distribution matrix* $\mathbf{D} \in \mathbb{R}^{m\times n}$, where m and n represent the numbers of HO-states and blocks, respectively. It distributes particles in each data block to corresponding HO-states. Formally, let two column vectors $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{s} \in \mathbb{R}^m$ represent the numbers of particles in blocks and HO-states, where elements b_i and s_j are the number of particles in the i-th block and the j-th HO-state, respectively. The distribution matrix $\mathbf{D}$ estimates $\mathbf{s}$ by $\mathbf{s} = \mathbf{D}\cdot\mathbf{b}$. Each element $D_{j,i}$ in the distribution matrix can be seen as the fraction of particles in block b_i that correspond to HO-state s_j (i.e., $s_j = b_i \cdot D_{j,i}$). The matrix $\mathbf{D}$ should fulfill two requirements: each column should be a partition of unity, meaning that the numbers of particles in HO-states should sum up to the number in the corresponding block, and each row should only have one nonzero entry, meaning that particles of an HO-state should only come from one corresponding block.

Aggregation. The second layer is the aggregation layer, represented by an *aggregation matrix* $\mathbf{A} \in \{0,1\}^{r\times m}$, where r denotes the number of HO-nodes. This layer aggregates the HO-states into HO-nodes to reduce the size of the network. Similarly, the matrix $\mathbf{A}$ estimates a column vector $\mathbf{n} \in \mathbb{R}^r$ representing the number of particles in each HO-node by computing $\mathbf{n} = \mathbf{A} \cdot \mathbf{s}$. The matrix $\mathbf{A}$ should only contain binary values. Specifically, an element $\mathbf{A}_{i,j}$ equals to one if the j-th HO-state is assigned to the i-th HO-node. Note that only the HO-states sharing the same current block (meaning that the particles are residing in the same block) could be aggregated into the same HO-node. Additionally, each column in $\mathbf{A}$ should be a one-hot vector, meaning an HO-state should only be assigned to a single HO-node.

Transition. The third layer is the transition layer, represented by the *probability transition matrix* $\mathbf{T} \in \mathbb{R}^{r\times r}$. This layer approximates the movement of particles between HO-nodes. We use the term "block steps" to denote the number of steps counted by blocks, reflecting the sequential movement of particles between blocks. Formally, given a column vector $\mathbf{n}^{(t)} \in \mathbb{R}^r$ representing the number of particles in each HO-node at block step t, the matrix $\mathbf{T}$ estimates the number at block step $t+1$ as $\mathbf{n}^{(t+1)} = \mathbf{T}\cdot\mathbf{n}^{(t)} \in \mathbb{R}^r$. Specifically, an element $\mathbf{T}_{i,j}$ represents the transition probability from the j-th HO-node to the i-th HO-node. Note that the transition is not always valid between any two HO-states. In our case, we enforce two constraints to avoid the violation of physical rules (e.g., a transition should not appear between two spatially disjoint blocks) and the violation of the semantic meaning of HO-states (e.g., a valid transition from state $A|B.C$ should move to states in the form of "$*|A.B$"). Accordingly, transitions between HO-nodes should be constrained as well. In practice, this constraint is enforced by a *mask matrix* $\mathbf{M} \in \mathbb{R}^{r\times r}$, where an element $\mathbf{M}_{i,j}$ indicates the validity of the transition from the j-th to i-th HO-nodes.

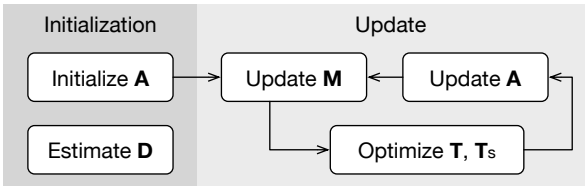


Fig. 3: Optimization routine for our FlowHON construction. The initialization stage initializes $\mathbf{A}$ and $\mathbf{D}$. The update stage iteratively optimizes $\mathbf{T}$ based on the current $\mathbf{A}$, and then updates $\mathbf{A}$ based on $\mathbf{T}$, so that the interaction between $\mathbf{A}$ and $\mathbf{T}$ is incorporated. The optimization is guided by a loss function based on $\mathbf{D}$, $\mathbf{A}$, and $\mathbf{T}$. The loss function is evaluated in the optimization and the validation, but they are hidden in this figure for visual compactness.

4.2 Optimization

The optimization routine for our FlowHON is illustrated in Figure 3. This routine aims to identify the optimal aggregation of HO-states and transitions between HO-nodes to best mimic the transition statistics in the sampled data. The optimization is performed in two stages: the initialization stage and the update stage. In the initialization stage, the initial values of the aggregation matrix $\mathbf{A}$ and the distribution matrix $\mathbf{D}$ are computed. These two matrices provide the initial HO-states and HO-nodes. In the update stage, a repeated procedure is performed to update the transition matrix $\mathbf{T}$ and the aggregation matrix $\mathbf{A}$ iteratively. In each iteration, we first optimize the transition matrix $\mathbf{T}$ based on the current HO-nodes ($\mathbf{A}$) and then update the HO-nodes ($\mathbf{A}$) based on the optimized transitions. The source code is at repository https://github.com/NanChanNN/FlowHON.

Loss function. The loss function evaluates the error of estimating the particle distribution over blocks using a model. This loss function is model-agnostic and can be applied to both first-order and higher-order networks. We use the estimation error to guide the optimization, as the estimation of particle movement is an essential task in graph-based flow visualization, on top of which many other applications

are built. Given the initial number of particles in blocks $\mathbf{b}^{(0)} \in \mathbb{R}^n$ at block step 0, we use the matrices $\mathbf{D}$, $\mathbf{A}$, $\mathbf{T}$ to estimate the numbers up to a predefined block step k (i.e., $\{\hat{\mathbf{b}}^{(t)} \in \mathbb{R}^n\}_{t=1}^k$). The estimated numbers are compared to the actual numbers from tracing (i.e., $\{\mathbf{b}^{(t)} \in \mathbb{R}^n\}_{t=1}^k$) using KL-Divergence (KLD). Each component in a vector $\mathbf{b}$ (or $\hat{\mathbf{b}}$) should be divided by the total number of particles to convert $\mathbf{b}$ (or $\hat{\mathbf{b}}$) to a probability distribution for the KLD computation. As this division only scales the loss by a constant factor, we keep using the vector $\mathbf{b}$ in our loss function for simplicity:

$$L = \sum_{t=1}^{k} \mathrm{d_{KL}}(\mathbf{b}^{(t)} || \hat{\mathbf{b}}^{(t)}) = \sum_{t=1}^{k} \sum_{i} b_i^{(t)} \cdot \log \frac{b_i^{(t)}}{\hat{b}_i^{(t)} + \epsilon}, \quad (1)$$

where $b_i^{(t)}$ and $\hat{b}_i^{(t)}$ are the actual number and the estimated number of particles in the i-th block at block step t, respectively, and ϵ is a small constant to avoid division by zero.

The vector $\hat{\mathbf{b}}^{(t)}$ is estimated using the matrices $\mathbf{D}$, $\mathbf{A}$, and $\mathbf{T}$. Given the initial particle number in blocks $\mathbf{b}^{(0)}$, the particle numbers in HO-states and in HO-nodes can be derived by $\mathbf{s}^{(0)} = \mathbf{D} \cdot \mathbf{b}^{(0)}$ and $\mathbf{n}^{(0)} = \mathbf{A} \cdot \mathbf{s}^{(0)} = \mathbf{A} \cdot \mathbf{D} \cdot \mathbf{b}^{(0)}$, respectively. Then, the movement of particles between HO-nodes can be approximated using the transition matrix $\mathbf{T}$, and the particle number in HO-nodes at block step t can be estimated as $\hat{\mathbf{n}}^{(t)} = \mathbf{T}^t \cdot \mathbf{n}^{(0)}$. Note that we use parentheses in superscripts to distinguish block steps (e.g., $^{(t)}$) from the power exponent (e.g., ^{t}). At each block step, we can aggregate the particle number in HO-nodes to the number in blocks using $\hat{\mathbf{b}}^{(t)} = \mathrm{nonzero}(\mathbf{D}^T \cdot \mathbf{A}^T) \cdot \hat{\mathbf{n}}^{(t)}$, where $\mathbf{D}^T$ and $\mathbf{A}^T$ are the transpose of $\mathbf{D}$ and $\mathbf{A}$, respectively, and $\mathrm{nonzero}(\cdot)$ is an element-wise function that sets nonzero elements in a matrix to 1. In summary, the estimated particle numbers in blocks $\hat{\mathbf{b}}^{(t)}$ at block step t is given by:

$$\hat{\mathbf{b}}^{(t)} = \mathrm{nonzero}(\mathbf{D}^T \cdot \mathbf{A}^T) \cdot \mathbf{T}^t \cdot (\mathbf{A} \cdot \mathbf{D} \cdot \mathbf{b}^{(0)}). \quad (2)$$

Therefore, we can see that the loss function only relies on $\mathbf{D}$, $\mathbf{A}$, and $\mathbf{T}$, which are the parameters to be optimized in our framework.

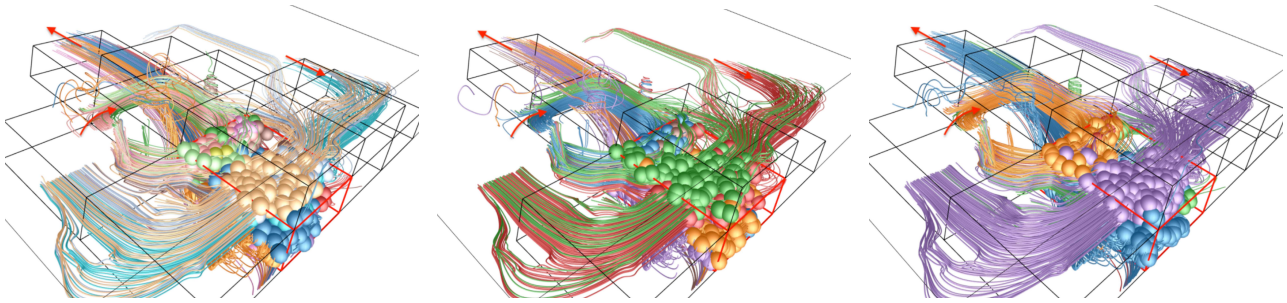
Estimate D. The construction of matrix $\mathbf{D}$ is part of the initialization phase. The matrix $\mathbf{D}$ transforms the number of particles in each block to the number of particles in each HO-state. We provide two strategies for the initialization: *exact assignment* and *approximate assignment*. The exact assignment uses backward tracing of particles to determine the exact visiting history. Given the visiting history, we could precisely assign a particle to its corresponding HO-state. While accurate, exact assignment suffers from the heavy computation of backward tracing for each particle. During experiments, we observed that the distribution from the blocks to the HO-states is quite stable at the beginning, which means that we could assign particles on a block to HO-states based on the statistics of previous samples. Motivated by this, we put forward the second strategy, namely approximate assignment. This strategy uses statistics from sampled particles to determine the fraction of particles corresponding to each HO-state in a block. The distribution matrix only reflects the statistics of HO-states when the visiting history is not given. As the distribution from the blocks to the HO-states is not involved in later transitions, this matrix will remain constant after initialization.

Initialize A (node generation). The aggregation matrix $\mathbf{A}$ has more columns than rows (i.e., $r < m$), which reduces the amount of HO-nodes in the resulted network. This matrix can be derived from any method that groups similar HO-states into HO-nodes. For example, we may use the variable-order network [38] to produce the aggregation matrix by considering the lower-order node to be a group of higher-order states. We call this method of node generation the *semantic* approach, since HO-states are clustered according to their semantic meanings. Specifically, an HO-state can be grouped into the respective lower-order node only if they share the same preceding history. However, this may fail to group HO-states with similar behavior but different previous blocks. In our implementation, we apply the *hierarchical clustering* to group similar HO-states in each block. The hierarchical clustering starts with clusters of individual HO-states and merges the two closest clusters in each iteration if their distance is smaller than a predefined threshold. The distance between two HO-states u and v is defined as the Euclidean distance d_E between their corresponding transition probability distributions p_u and p_v. The distance $d(\mathbf{c}_i, \mathbf{c}_j)$ between the two clusters $\mathbf{c}_i$ and $\mathbf{c}_j$ is defined as the weighted average of distances between their HO-states:

$$d(\mathbf{c}_i, \mathbf{c}_j) = \frac{\sum_{u \in \mathbf{c}_i, v \in \mathbf{c}_j} w_u \cdot w_v \cdot \mathrm{d_E}(p_u, p_v)}{\sum_{u \in \mathbf{c}_i} w_u \cdot \sum_{v \in \mathbf{c}_j} w_v}, \quad (3)$$

where w_u and w_v are the amounts of transitions related to HO-states u and v, respectively. Note that we use transition probabilities from HO-states to blocks, not between HO-states, to avoid clustering overfitting.

Figure 4 compares individual HO-states, HO-nodes generated by the semantic scheme, and HO-nodes generated by hierarchical clustering. The spheres represent particles within the block highlighted in red, with their colors indicating the corresponding HO-nodes. The streamline segments corresponding to the particles are depicted in the same respective color. We observe that particles of the same color are located in similar regions within the block. These regions irregularly partition the block, with streamlines in the same area exhibiting similar behaviors. Ideally, our goal is to represent all streamlines with identical patterns using a single HO-node, thereby avoiding redundancy. This means that streamlines sharing the same pattern should be depicted in the same color. In Figure 4 (a), individual HO-states depict flow patterns at the finest level. However, multiple HO-states often represent streamlines with similar patterns, resulting in a mix of colors and redundancy. The semantic scheme, as shown in Figure 4 (b), attempts to group HO-states into HO-nodes according to predefined semantic rules. However, this method is limited by its reliance on semantics and does not fully achieve the desired outcome. For example, some green particles share a transition pattern with the red ones, while others align more closely with the blue particles. In contrast, Figure 4 (c) illustrates that HO-nodes generated through hierarchical clustering provide a more concise summarization of transition patterns. Particles with the same pattern are consistently grouped together, sharing the same color and moving towards the same block,



(a) (b) (c)

Fig. 4: Visualization of HO-nodes produced by different schemes to group HO-states. (a) shows the individual HO-states. (b) shows the HO-nodes generated by the variable-order network [38], or the semantic scheme. (c) shows the HO-nodes generated by hierarchical clustering. The red box indicates the selected block, and the red arrows indicate the flow directions. The spheres represent sampled particles, with colors corresponding to respective HO-nodes. In each subfigure, spheres sharing the same color belong to the same HO-node (or HO-state).

resulting in a clearer and more effective representation of flow behaviors.

Update M. An element $M_{i,j}$ in the mask matrix $\mathbf{M}$ records whether the transition between the i-th and j-th HO-nodes is physically meaningful. Given the aggregation matrix $\mathbf{A}$, this can be easily done by checking whether the transition between any pair of HO-states is possible. In practice, we may count the number of transitions between two HO-nodes in the sampled data, setting the corresponding element in $\mathbf{M}$ to zero if the count is zero. An zero entity in $\mathbf{M}$ prevents the transition probability between corresponding HO-nodes from being updated during transition matrix optimization. The mask matrix update is involved in the update loop in Figure 3 as it relies on the aggregation matrix $\mathbf{A}$. Therefore, once $\mathbf{A}$ is updated, the mask matrix $\mathbf{M}$ will also be updated.

Optimize T (edge optimization). The transition matrix T encodes the edges among HO-nodes in a HON. Traditional approaches [31], [38], [39] count the transitions between HO-nodes and normalize the count into the probability. However, these approaches do not consider the dynamic patterns of transitions, and therefore may overemphasize the transition patterns when a block contains a large number of particles. To circumvent this problem, rather than relying on statistics collected from sampled particles, we aim to learn a transition matrix $\mathbf{T}$ that can produce the observed particle distributions at each block step. We start with a matrix $\mathbf{T}^{(0)}$ from sampled particles and update the matrix to minimize the loss function L (Equation 1) using the gradient descent algorithm. Formally, the update process is performed iteratively using the following equation:

$$\mathbf{T}^{(i+1)} = \mathbf{T}^{(i)} + \alpha \frac{\partial L}{\partial \mathbf{T}^{(i)}} \odot \mathbf{M}, \tag{4}$$

where α is the learning rate and $\odot$ denotes element-wise multiplication. For simplicity, we use $\mathrm{nonzero}(\mathbf{T}^{(0)})$ as the mask matrix $\mathbf{M}$. For $\mathbf{T}$ to be a transition probability matrix, two additional constraints are enforced. First, all elements in $\mathbf{T}$ should be non-negative. We utilize projected gradient descent [19] to replace all negative values by zero after every weight update. Second, the summation of elements in each column in $\mathbf{T}$ must equal to one for this column to be a distribution. We enforce this by adding a penalty term in the loss function to punish columns whose element summations deviate from one:

$$L_T = L + \sum_j (\sum_i T_{i,j} - 1)^2. \tag{5}$$

Besides, we normalize the transition matrix at the end of the optimization process to make the summation of each column strictly equal to one. While the normalization process guarantees the partition of unity, the least-squares penalty term (Equation 5) is still necessary to ensure that the norm of distribution is not too close to zero during optimization, which could cause unstable numerical issues.

Update A (node update). Note that an HO-state should be aggregated into the HO-node with the most similar transition behavior. Therefore, when the transition probability ($\mathbf{T}$) of HO-nodes is updated, the aggregation from HO-states to HO-nodes should be updated accordingly. Because the transitions from the HO-states to HO-nodes are unknown, we use optimization to learn a transition matrix $\mathbf{T}_s$ using a similar scheme as the $\mathbf{T}$ update. Note that each column in $\mathbf{T}_s$ contains the transition probabilities from a HO-state to all HO-nodes, and each column in $\mathbf{T}$ contains the probabilities from a HO-node to all HO-nodes. Therefore, for each HO-state, we can identify the most similar HO-node by comparing the corresponding columns in $\mathbf{T}_s$ and $\mathbf{T}$ using Euclidean distance. The aggregation matrix $\mathbf{A}$ is constructed so that each HO-state is assigned to the most similar HO-node.

Termination. We terminate the optimization routine if one of the following two criteria is fulfilled. First, the node assignment does not change for a predefined number of iterations, which is done by checking whether any value in $\mathbf{A}$ has changed during the update of $\mathbf{A}$. Second, the performance of the model does not improve for a predefined number of iterations. We use a validation set to evaluate the performance and pick the model with the minimum validation loss for later use.

5 Results and Discussion

5.1 Experiment Configuration

In the experiment, we compare our approach with the traditional first-order network (denoted as FON) and the

TABLE 1: Summary of data sets and timing performance. Block dimensions represent the number of blocks along each axis. The "init" and "train" columns show the node initialization and transition optimization time, respectively. The "ours-3rd", "ours-2nd", and "ours-4th" columns reflect the construction time for the third-order, second-order, and fourth-order FlowHON, respectively. For FON, VAR, and Fixed, "init" specifies the original method's construction time, while "init + train" indicates the total time for the optimized version. The time is measured in seconds.

data set	data dimension	block dimension	FON		VAR		Fixed		ours-3rd		ours-2nd		ours-4th	
			init	train	init	train	init	train	init	train	init	train	init	train
ABC	$64 \times 64 \times 64$	$6 \times 6 \times 6$	0.48	5.93	0.72	27.91	0.32	98.22	3.18	20.22	0.17	13.13	10.82	27.13
Bénard	$128 \times 32 \times 64$	$8 \times 4 \times 4$	1.82	3.50	3.04	14.08	0.82	84.00	10.34	51.01	0.37	11.98	60.92	75.26
combustion	$506 \times 400 \times 100$	$10 \times 8 \times 2$	0.45	5.07	0.63	21.59	0.43	70.74	6.58	22.71	0.12	6.31	145.61	206.91
computer room	$417 \times 345 \times 60$	$6 \times 6 \times 3$	0.30	3.46	0.43	8.91	0.34	31.66	3.81	13.29	0.16	5.73	31.56	53.67
crayfish	$322 \times 162 \times 119$	$10 \times 5 \times 4$	0.73	5.29	1.15	44.59	0.87	183.22	13.72	36.52	0.54	8.65	210.79	283.32
electron	$64 \times 64 \times 64$	$5 \times 5 \times 5$	0.25	4.96	0.29	6.66	0.17	16.62	2.34	10.99	0.13	5.16	20.11	31.76
five critical points	$51 \times 51 \times 51$	$4 \times 4 \times 4$	0.24	3.37	0.30	4.18	0.15	10.28	1.24	7.52	0.05	5.35	5.32	66.62
hurricane	$500 \times 500 \times 100$	$6 \times 6 \times 2$	0.44	3.63	0.65	7.77	0.19	20.98	1.40	22.56	0.07	8.20	10.62	81.13
solar plume	$126 \times 126 \times 512$	$4 \times 4 \times 10$	1.37	5.37	1.30	21.45	0.79	88.90	10.89	34.15	0.24	7.18	132.19	175.31
square cylinder	$192 \times 64 \times 48$	$10 \times 3 \times 2$	0.42	3.46	0.56	3.87	0.07	6.06	0.26	16.29	0.02	5.35	1.78	15.54
tornado	$64 \times 64 \times 64$	$5 \times 5 \times 5$	0.41	3.54	0.63	6.15	0.09	12.66	0.47	7.30	0.05	8.58	1.76	43.27
two swirls	$64 \times 64 \times 64$	$5 \times 5 \times 5$	1.33	3.84	2.23	19.94	0.53	88.11	8.60	67.18	0.30	34.54	85.46	103.23
average	/	/	0.69	4.29	0.99	15.59	0.40	59.29	5.24	25.81	0.19	10.01	59.75	96.93

variable-order network (denoted as VAR) [38]. The fixed-order network (denoted as Fixed) [31] can be considered as a reference because it represents the finest level of transitions with the largest number of nodes. Both VAR and FlowHON are approximating the behavior of Fixed with fewer nodes. Since FON, VAR, and Fixed can be considered as special solutions of $\mathbf{D}$, $\mathbf{A}$, and $\mathbf{T}$, they can all benefit from our transition optimization that updates $\mathbf{T}$. Therefore, we additionally evaluate the optimized variants of FON, VAR, and Fixed, referred to as FON+, VAR+, and Fixed+, respectively. Specifically, we apply the transition optimization methods described in Section 4.2 to their transition matrix $\mathbf{T}$ while keeping their $\mathbf{D}$ and $\mathbf{A}$ unchanged.

Tasks. We evaluate the performance of these approaches on three tasks: namely, *particle density estimation*, *community detection*, and *graph visualization*. The particle density estimation starts from a set of uniformly sampled particles and uses the networks to estimate the number of particles over blocks at each block step. This task quantitatively evaluates the ability of a network to approximate the transition patterns between blocks. The community detection partitions the flow field into communities and examines the average number of times a particle moves between two communities. This task quantitatively evaluates the effectiveness of data partition based on different networks. A network is favored if a particle is less likely to move between different communities. This indicates that the communities are more independent and that less task exchange or data loading is needed in parallel particle tracing. Finally, we visualize the networks using node-link diagrams to evaluate them qualitatively, aiming to see what kind of structural information can be revealed by different networks.

Data sets and training configuration. We use twelve steady flow data sets with different characteristics in the experiment and summarize the data statistics and model construction times in Table 1. The experiment with these data sets are described in Section 5.2, Section 5.3, and Section 5.4. We further include preliminary experiments with two unsteady flow data sets, which will be discussed in Section 5.5. For the steady fields, we select the block dimension proportional to the data dimension so that each block is roughly a cube. For each data set, we sample 15,000 particles for the network construction (10,000 are used for training and 5,000 for validation) and 15,000 for testing. We only use forward tracing to produce streamlines to avoid all streamlines starting from boundary blocks. The streamlines are then converted into a sequence of blocks. A special block "-1" denotes the end of streamlines, either due to going out of boundaries or reaching critical points. For the node generation, we use a difference threshold of 0.04 for hierarchical clustering. We train each transition matrix in 100 epochs and set the learning rate to 0.01 with a decay rate of 0.9 every 10 iterations. In our implementation, we use TensorFlow to compute the derivatives and update the transition matrix. We terminate the optimization if the matrix $\mathbf{A}$ does not change for 4 consecutive iterations. More details on the experimental setup, including dataset statistics, threshold selection analysis, convergence testing of transition matrix training, and impact of different block dimension settings, are provided in Appendix B.

Construction time The third-order FlowHON construction requires 5.24 seconds for node initialization and 25.81 seconds for edge optimization, leading to a total construction time of 31.05 seconds on average, as shown in Table 1. The construction time ranges from 7.77 seconds for the tornado data set to 75.78 seconds for the two swirls data set. Compared with the other approaches, it requires the most time in the initialization stage, as an additional hierarchical clustering is used. FON (0.69 seconds) and Fixed (0.40 seconds) require the least time to initialize, as they only connect the nodes based on statistics without any further analysis. In terms of edge optimization, FON (4.29 seconds) is the fastest with the smallest number of nodes, and Fixed (59.29 seconds) is the slowest with the most number of nodes. FlowHON (25.81 seconds) is slower than VAR (15.59 seconds) with similar amount of nodes. VAR does not update the node aggregation and, therefore, does not benefit from the iterative optimization scheme. Instead, VAR performs a single iteration of optimization, leading to a faster edge training time.

5.2 Particle Density Estimation

Experiment setup. For this task, we compute the three matrices $\mathbf{D}$, $\mathbf{A}$, and $\mathbf{T}$ for each network and use Equation (2)

TABLE 2: Average estimation error and network sizes of different network construction approaches. The estimation error is given by KLD using Equation 1. 'VAR' denotes the variable-order network up to third-order, and 'Fixed' denotes the third-order fixed-order network. The smallest estimation error for each data set among the third-order FlowHON, FON, VAR, FON+, and VAR+ is highlighted in bold font. The network size denotes the number of nodes.

	estimation error									network size					
	other techniques			with edge optimization			ours			other techniques			ours		
data set	FON	VAR	Fixed	FON+	VAR+	Fixed+	3rd	2nd	4th	FON	VAR	Fixed	3rd	2nd	4th
ABC	0.029	0.022	0.013	0.029	0.019	0.012	**0.016**	0.019	0.016	216	978	2180	641	469	735
Bénard	0.057	0.062	0.055	0.022	0.017	0.014	**0.016**	0.017	0.016	128	968	2260	412	361	556
combustion	0.024	0.021	0.017	0.022	**0.014**	0.014	0.017	0.018	0.016	141	761	1879	593	376	798
computer room	0.028	0.026	0.012	0.023	0.015	0.011	**0.014**	0.015	0.012	107	698	1631	593	348	747
crayfish	0.091	0.084	0.054	0.027	0.019	0.014	**0.019**	0.019	0.019	200	1486	3249	796	562	973
electron	0.006	0.008	0.006	0.007	0.008	0.007	**0.006**	0.007	0.006	125	348	1018	330	263	370
5 critical points	0.016	0.015	0.008	0.011	0.010	0.007	**0.008**	0.009	0.006	64	272	751	296	214	360
hurricane	0.030	0.037	0.017	0.019	0.015	0.010	**0.010**	0.013	0.011	72	354	880	289	247	393
solar plume	0.122	0.115	0.082	0.074	**0.045**	0.029	0.051	0.053	0.051	160	1009	2252	606	410	738
square cylinder	0.006	0.006	0.002	0.005	0.002	0.002	**0.002**	0.004	0.001	60	185	384	183	104	227
tornado	0.014	0.016	0.011	0.015	0.011	0.012	**0.011**	0.012	0.010	125	348	754	309	259	357
two swirls	0.077	0.059	0.052	0.026	0.019	0.019	**0.019**	0.021	0.016	125	904	2041	404	287	594
average	0.042	0.039	0.028	0.023	0.016	0.013	**0.016**	0.017	0.015	127	693	1607	454	325	570

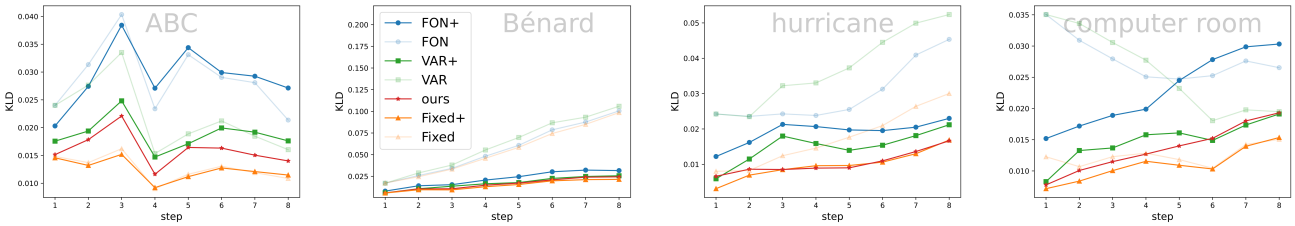


Fig. 5: Estimation error of particle density (y-axis) over block steps (x-axis) for four data sets. The blue, green, orange, and red curves show the accuracy of FON, VAR, Fixed, and FlowHON, respectively. Lighter-colored curves indicate the estimation error of the respective original networks without edge optimization, while darker-colored curves show the estimation error after applying our edge optimization. More contents are available in Appendix C.

to estimate the number of particles over blocks at each block step. We use Equation (1) to calculate the estimation error for the first eight block steps. The reported error is divided by the number of particles, which converts the vector $\mathbf{b}$ back to a probability distribution and normalizes the loss. The initial positions of particles are evenly distributed in space for all training, validation, and test data. The ground-truth is obtained by tracing particles with the fourth-order Runge-Kutta method and counting the particles in each data block. Table 2 shows each approach's average estimation error and network size for each data set.

Impact of the order of network. We first compare the networks of different orders using FlowHON. In general, the estimation error decreases slightly and network size increases with the increase of order. The third-order network outperforms the second-order one for all data sets, and the fourth-order network outperforms the third-order one for most of the data sets, except for the hurricane data set. But, in general, we find that the errors are similar across different orders. To balance between the error and the size of the network, we will use the third-order network for comparison among different approaches.

Comparison with previous approaches. Our FlowHON has smaller estimation errors than FON and VAR for all data sets. The average error of FlowHON (0.016) is more than 50% smaller than the average of FON (0.042) and VAR (0.039). FlowHON also outperforms Fixed on most data sets (ten out of twelve), except the ABC and computer room. But the average improvement is smaller, with the average error of Fixed being 0.028. It should be noted that the Fixed achieves similar errors to FlowHON on many data sets, and the difference of average errors majorly comes from a few data sets, such as the Bénard, crayfish, solar plume, and two swirls. For these data sets, Fixed has errors larger than 0.05. For the solar plume data set, FlowHON also has an error of 0.051, but for all the other data sets, the errors are smaller than 0.02. In terms of the size, FON has the smallest number of nodes. For HON approaches, FlowHON is the smallest on average (454.3), which is 34.4% smaller than VAR (692.6) and 71.7% than Fixed (1606.6).

Impact of our edge optimization. All existing approaches benefit from our transition optimization procedure in most cases. On average, the estimation errors reduce by 26.6% for FON, 45.9% for VAR, and 30.6% for Fixed. The small size of FON may limit the power to precisely describe the transition patterns, leading to the smallest improvement with our optimization. On the contrary, Fixed may already capture most transition patterns and has less room to improve, compared with VAR. After the optimization, FON still has the largest average error (0.023), and Fixed has the smallest error (0.013). FlowHON (0.016) and VAR (0.016) have similar errors, which are close to that of Fixed. But FlowHON is 34.4% smaller in size and achieves smaller errors on ten of the twelve data sets. Overall, we observe that edge optimization is more crucial than the initialization of nodes, and network size limits the enhancement achieved.

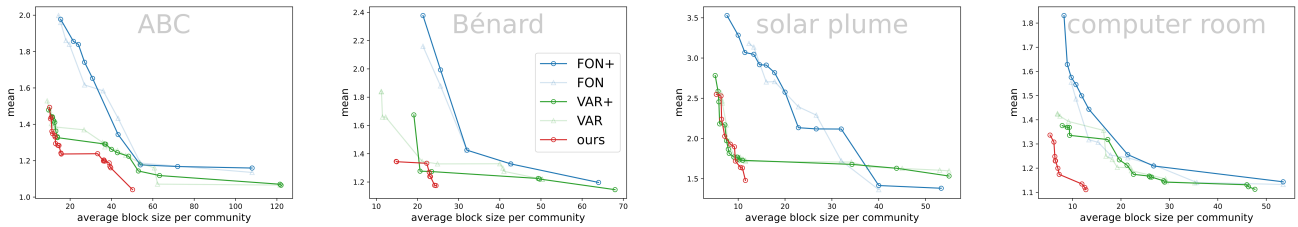


Fig. 6: Community detection results on four data sets. The x-axis denotes the average community size regarding the number of data blocks, and the y-axis denotes the mean of community visits. For the same community size, smaller means of community visits are preferred. Blue, green, and red curves denote the results of FON, VAR, and FlowHON, respectively. Curves with lighter colors show the result of respective original networks without edge optimization, and curves with darker colors show that of optimized networks. More contents are available in Appendix C.

While it may not significantly benefit certain datasets, such as electron and tornado, it proves effective for most cases. Note that developing a method to predict the impact of edge optimization on the accuracy without actually performing it is a promising direction for future work. Figure 5 shows the error estimation over block steps for parts of the data sets. We find that the estimation errors usually follow similar patterns for all approaches, except those without edge optimization (lighter-colored curves). We can confirm that Fixed with our edge optimization usually delivers the best accuracy, but ours is often close. This implies that existing graph-based algorithms may be equipped with our network for better performance. For example, random walks can simulate the movement of particles in our network to deliver better time performance than Fixed and more accurate estimation results than VAR and FON.

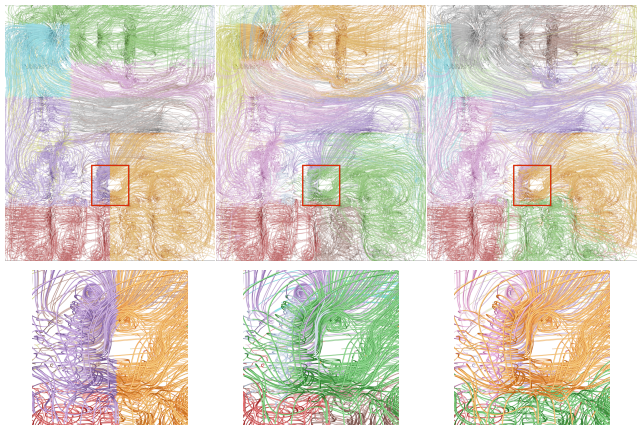
5.3 Community detection

Experiment setup. In parallel particle tracing, data-parallelism and task-parallelism require effective flow field partition to reduce the amount of particle exchanging or data loading [45]. In this task, we apply a community detection algorithm on FlowHON to examine its effectiveness in guiding flow field partition. During community detection, HO-nodes in the network are partitioned into communities, with nodes in the same community depicting similar flow patterns. This partitioning can guide parallel particle tracing: in task-parallelism, each computing node will handle a single community, while in data-parallelism, communities will be loaded sequentially to the computing node as needed. The actual data loaded during tracing consists of the data blocks associated with the HO-nodes in a community. To evaluate the partitioning results, we trace a set of particles and measure two metrics. The first is the mean of community visits, which records how often a particle transitions between different communities. Specifically, we track the movement of a particle on HO-nodes and increase the count when it moves from its current community to a different one, including a previously visited one. A count of one indicates that the particle remains within a single community throughout tracing. This metric roughly estimates the number of particle exchanges required in task-parallelism, as each inter-community transition involves communication between computing nodes. Similarly, it indicates the number of data-loading operations needed when data-parallelism is employed, as transitioning to a new community requires loading its associated data blocks. The second metric is the average community size, calculated as the average number of data blocks per community. This metric reflects the average amount of data to be loaded for each computing node in both task- and data-parallelism.

We use InfoMap [30] to identify communities in the network. It controls the resulting community resolution with a parameter *Markov-time*. Generally, a higher *Markov-time* results in a smaller number of communities with larger sizes. To avoid trial-and-error effort to select this parameter, we evenly sample the parameter from 0.5 to 3.5 with a step of 0.1, and record the resulting average community sizes and the mean of community visits using a sample set of particles. The parameter values at the Pareto front are used in the testing stage with new sets of particles. We compare our approach with FON+ and VAR+. We do not compare with Fixed+ as that may produce many small communities; each contains a bundle of streamlines.

Community detection results. Figure 6 shows the average community sizes (in data blocks) versus the mean of community visits for four data sets based on the test set of particles. Given the resource limits, a preferred parameter value should lead to the smallest mean of community visits with an affordable community size. In Figure 6, we find that all curves decrease monotonically when the average community size increases. This indicates that the parameter values collected from the sampling particles can provide useful hints for the testing particles as well. Compared to FON+ and VAR+, FlowHON produces smaller means of community visits in most cases. An exception is the solar plume data set, for which our FlowHON and VAR+ perform similarly for smaller community sizes. But our FlowHON rarely produces large communities, while VAR+ may produce much larger communities with a marginal decrease in the mean of community visits. Additionally, the HON approaches (i.e., FlowHON and VAR+) outperforms FON for almost all cases.

Figure 7 illustrates an example of the data partitioning results from different methods, all of which produce a similar performance in terms of the mean of community visits. In HON approaches, communities are detected at the HO-node level, which means that a data block can belong to multiple communities, leading to overlapping subregions with different colors within the same block. We



(a) (b) (c)

Fig. 7: Data partition based on community detection results for the computer room data set. In each subfigure, we track the movement of a particle (represented by a streamline) across HO-nodes and color the corresponding streamline segment according to the community the particle is currently passing through, with each community assigned a unique color. Consequently, if a particle moves through multiple communities, its streamline may exhibit multiple colors. The top row displays the partition of the entire field, while the bottom row zooms in on the regions highlighted by the red rectangles. Columns (a)–(c) present the results from FON+, VAR+, and FlowHON, respectively. Note that colors do not correspond directly across multiple subfigures.

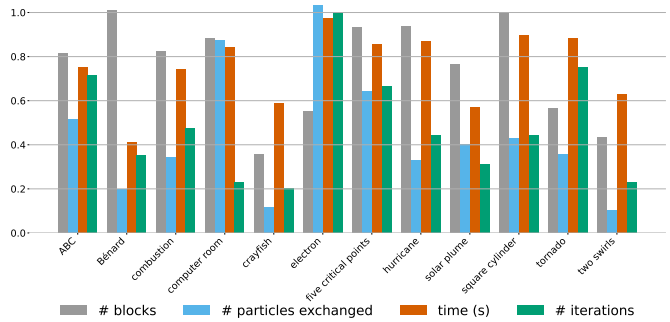


Fig. 8: Comparison of the parallel particle tracing performance delivered by the FlowHON communities and FON communities. The gray bars are the ratios of the average sizes of FlowHON communities (in terms of the number of blocks loaded) over those of FON communities. The blue bars show the ratios of the numbers of particles exchanged using FlowHON over those using FON. The red bars represent the ratios of timing (in seconds) using FlowHON over that using FON. The green bars indicate the ratios of the numbers of iterations required using FlowHON over those using FON. For each metric, a bar lower than 1 indicates that FlowHON outperforms FON, with a lower bar signifying a better performance by FlowHON.

observe that HON approaches generally produce smaller and overlapping communities. This is supported by quantitative results: FlowHON yields an average community size of 5.37 with a mean of community visits of 1.37; VAR+ generates an average size of 6.89 with a mean of visits of 1.43; and FON results in larger communities with an average size of 9.72 and a mean of visits of 1.55. It is important to note that smaller and overlapping communities may require more computing nodes but demand fewer resources per node, making them more compatible with modern parallel computing hardware. Furthermore, HON's ability to subdivide blocks into irregular subregions allows it to create "soft" boundaries of communities, by assigning particles of different behaviors within boundary blocks to different communities. These "soft" boundaries enable a more natural partitioning of the flow field, adapting to flow patterns rather than being constrained by predefined block partitions. This is evident in Figure 7 (b) and (c), where flows with similar patterns are depicted with consistent coloring. In contrast, FON, limited by regular block partitioning, can only produce "hard" boundaries among blocks that may unnaturally separate a flow feature. This limitation is clearly visible in the magnified region of Figure 7 (a), where the flow is split between the purple and orange communities, creating an artificial partition boundary.

Parallel particle tracing. We implement a simple parallel particle tracing strategy using MPI to examine the performance delivered by the traditional FON communities and FlowHON communities. With this strategy, each computing node loads the data blocks corresponding to one community and processes the particles in that community. The computation is performed in multiple iterations. In each iteration, a computing node traces every particle assigned to it until the tracing is finished or the particle goes out of the community. After each iteration, all computing nodes exchange the particles that are not completely traced. The particles are sent to a destination computing node based on its HO-states. The iteration repeats until all particles are fully traced. In this experiment, we trace 1,000,000 particles for each flow field and run the parallel tracing program on 750 processes within a CPU cluster of 50 computing nodes, where each node has an Intel Xeon E5-2692 CPU running at 2.2GHz with 64GB memory. As the number of processes is larger than that of communities, we duplicate a community for multiple computing nodes in practice, ensuring that FON and FlowHON consume similar computation resources for a fair comparison. Figure 8 shows the comparison in four aspects: the average number of blocks loaded by computing nodes, the number of particles exchanged during tracing, the total tracing time, and the number of iterations required to complete the tracing process. As these four numbers vary significantly across data sets, we use ratios between the numbers of FlowHON and those of FON for easier comparison. Here, the average number of blocks is measured from the actual loading of computing nodes, which is different from the average community size in Figure 6.

On average, FlowHON, when compared to FON, requires 44.5% of the number of particles exchanged during tracing and 48.5% of the number of iterations needed, leading to a 75.1% of the whole tracing time under the same computation resources. At the same time, FlowHON loads fewer blocks to the computing nodes and incurs less memory burden. However, we should note that the naive parallel particle tracing does not fully leverage the power of FlowHON. On average, the reduction in run time (24.9%) is not as significant as the reduction in the number of particles

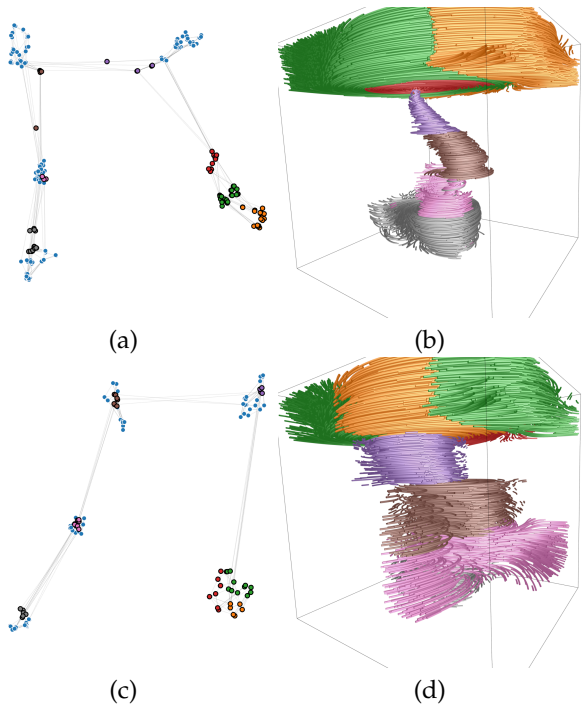


Fig. 9: Exploring the tornado data set using FlowHON (first row) and FON (second row). The streamline segments corresponding to the selected nodes are shown. In subfigures (a) and (c), the light blue spheres represent the nodes that have not been selected.

exchanged (55.5%) and the number of iterations (51.5%). This is due to the unbalanced workload across computing nodes and iterations. In later iterations, several particles traveling between a few computing nodes may delay the entire computation. As the total amount of computation is constant, the inefficient execution in these iterations reduces the overall computation power utilization, leading to a longer execution time. A more sophisticated parallel particle tracing algorithm may avoid this by balancing the workload, and FlowHON can also be beneficial in this aspect for two reasons. First, FlowHON provides a more accurate estimation of the particle densities, as shown in Table 2. This can give a more precise estimation of the number of particles processed by each computing node. Second, FlowHON distinguishes different patterns inside a block. As the trajectories of particles may vary significantly in length, FlowHON may facilitate a more accurate estimation of the amount of computation required by different kinds of particles. The improvement in these two aspects may boost the performance of workload estimation, and, therefore, enhance the workload allocation.

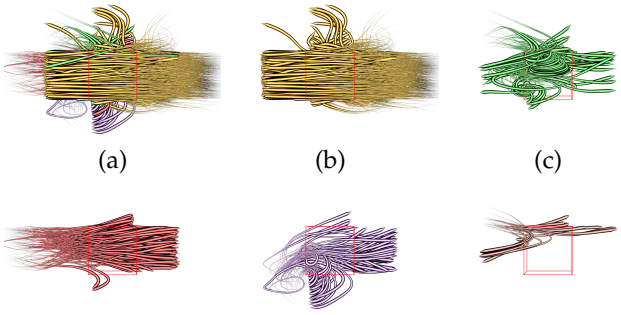
5.4 Visual Exploration

We further examine the constructed network utilizing the graph layout derived from the LinLog energy model, a widely adopted layout algorithm [24]. We implement a simple exploration system that supports brushing and linking between the graph visualization and the streamline visualization. Once a node is selected in the graph visualization, the corresponding streamline segments related to that node are visualized in the same color.

Tornado. The tornado data set is divided into $5 \times 5 \times 5$ blocks. In Figure 9 (a) and (c), we find that both FlowHON and FON exhibit five groups of nodes, corresponding to the five vertical layers of the data set. Additionally, for FlowHON, most of the five groups of nodes demonstrate a finer level of structures. For example, the group of nodes at the bottom-right corner can be easily divided into three smaller groups, as shown in orange, green, and red in Figure 9 (a). Their corresponding streamline segments locate at the top layer of the data set, where the segments related to the red nodes occupy the central region, as shown in Figure 9 (b). For other groups, we can also observe some nodes that are visually separable from other nodes, as colored in purple, brown, pink, and gray. These nodes correspond to streamlines at the core of the tornado as well. This shows that FlowHON captures the core of the tornado. The network produced by FlowHON is consistent with our understanding of the data set: only the particles around the core of the tornado will move across vertical block layers, while the particles at the outer layer of the tornado will mostly move horizontally. The particle transitions along the vertical core enhance the connections among HO-nodes corresponding to the core of the tornado and drag them away from other nodes at the same vertical layer. In contrast, FON, constrained by rigid block partitioning, struggles to distinguish different structures in the flow field. This is most obvious for the top layer of blocks, where the corresponding nodes in FON shows a single group with mixed structures in the bottom-right corner of (c), while the nodes in FlowHON form three groups distinguishing different layers of flow in (a). This lack of distinction makes it more challenging to interpret the flow field and accurately identify finer-grained structures from the graph layout. Furthermore, structures, such as the tornado core, may not align with block boundaries, resulting in the core being less distinctly revealed compared to FlowHON. Consequently, FON provides a less detailed and structured summarization of flow dynamics. Due to space constraints, we delegate the visual exploration of another data set (solar plume) to Appendix D for reference.

5.5 Preliminary Experiment with Unsteady Flows

Experiment setup. We experiment with FlowHON on two unsteady flow data sets: the hurricane data set, and the European Center for Medium Range Weather Forecasts (ECMWF) data set. The hurricane data set contains 48 time steps (one per hour) over the Mexico Gulf, while the ECMWF data set contains 44 time steps (one per month) over the earth. Therefore, the hurricane data set depicts a relatively short-term atmospherical flow in a local region, while the ECMWF data set describes a long-term global unsteady flow. For each time step, we generate 1,000 pathlines starting from that time step. For the ECMWF data set, we use a very small time interval between tracing steps as the interval between the time steps is relatively large. Therefore, the pathline may be similar to streamlines traced at individual time steps for this data set. The pathlines are first converted into sequences of blocks. Then, the networks



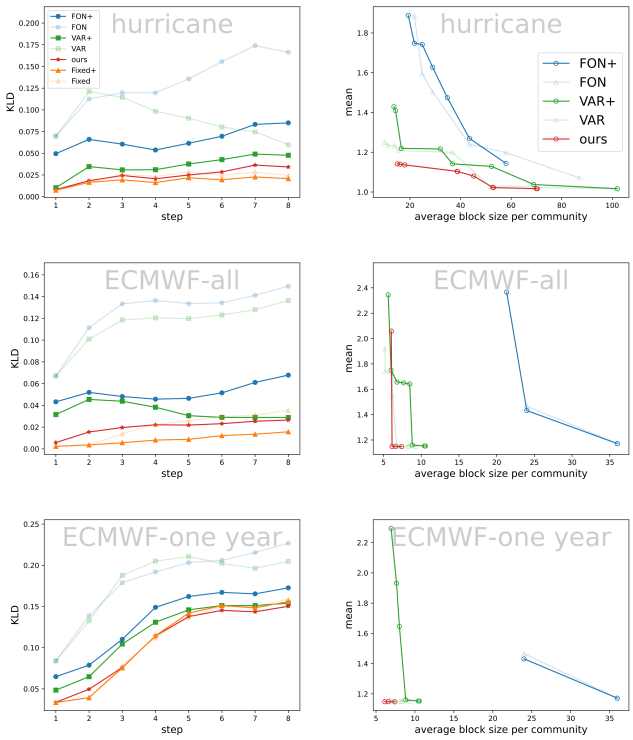
(a) (b) (c) (d) (e) (f)

Fig. 10: Flow patterns related to a single block in the unsteady ECMWF data set. (a) shows the HO-nodes contained in the block highlighted in the red box. (b) to (f) show the pathlines related to the five major HO-nodes in the block, respectively. HO-nodes are generated by FlowHON, and the particles flow from the narrower end to the broader end.

are constructed from the block sequences using exactly the same scheme as that for steady flows.

Exploration results. For unsteady flow fields, the trajectories of particles inside each data block may be even more diversified. For clarity, we illustrate the direction of pathlines to show that particles move from the narrower end to the broader end. Figure 10 (a) shows the HO-nodes contained in the selected block in red, while (b) to (d) depicts the pathlines corresponding to five major HO-nodes. The HO-nodes are listed in the order of the number of pathlines related to them. The orange and red pathlines represent two patterns that are similar in shape but move in opposite directions. The green group corresponds to spirals going outward from the selected block, while the purple group corresponds to spirals moving between the selected block and the neighbor on the south. The brown group represents pathlines that split into two branches after entering the block. These findings underscore the value of higher-order dependencies in detecting flow patterns, supporting the efficacy of FlowHON to represent flow fields.

Quantitative results. Figure 11 compares the performance of different networks on particle density estimation and community detection tasks. The first two rows show results using all time steps for training and testing. In terms of the density estimation, we find that networks with edge optimization clearly outperform their original versions, especially for FON and VAR. For Fixed, the edge optimization does not lead to an obvious improvement. This reveals that the finest-level HO-states play a major role in the estimation, while the optimization only recovers the information on a simplified network with a reduced number of nodes. For unsteady flows, we find that FlowHON outperforms VAR+ and is very close to Fixed+. This demonstrates the effectiveness of our node initialization and iterative update scheme. As the ECMWF data set contains many periodic flow patterns over the years, we further examine whether data from previous years can be used to predict the transitions in later years. From the third row, we find that the edge optimization still brings a performance gain, but the estimation error clearly increases as the tracing goes on for all approaches. This may indicate that the difference



(a) (b)

Fig. 11: Particle density estimation (a) and the community detection (b) results for the unsteady flow fields. The first row shows the results of the hurricane data set using all time steps. The second row shows the results of the ECMWF data set using all time steps for training and testing. The third row shows the results of the ECMWF data set using one year (12 time steps) for training and two years (24 time steps) for testing.

between transition patterns of different years requires a more sophisticated optimization technique. Regarding the community detection result, we find that FlowHON outperforms FON+ and VAR+ as the particles visit fewer communities on average with smaller community sizes.

In Appendix E, we include an alternative version of the visual exploration results with geo-reference and experiment on more unsteady flow fields where we partition the unsteady flow field into space-time blocks. Results are consistent with those discussed in this section.

6 Conclusions and Future Work

We propose FlowHON that describes particle transitions at the block level using higher-order networks. We formulate the higher-order network construction as an optimization problem of three linear layers, corresponding to the distribution from blocks to higher-order states, the aggregation from higher-order states to higher-order nodes, and the transitions among higher-order nodes. Higher-order nodes subdivide the flow behaviors within individual blocks, leading to better accuracy in describing particle transitions among blocks. The higher-order network connects the higher-order nodes and describes the transition patterns at a larger scale. We assess our approach's effectiveness by comparing it with

existing graph-based approaches from multiple aspects. We experiment with both steady and unsteady flow fields on three downstream tasks. Results show that FlowHON outperforms existing approaches in most cases, and our edge optimization approach could also help existing approaches enhance their performance.

In the future, we would like to explore the following directions. First, we would like to develop scalable parallel particle tracing techniques on top of FlowHON. Our simple parallel particle tracing platform with FlowHON has demonstrated the potential of FlowHON in reducing particle exchanges and tracing iterations. But workload balance in the platform remains an open question. We would like to further explore whether the higher-order nodes will provide more accurate information to estimate the workload in each community as well. Based on the accurate transition pattern and workload estimation, we may be able to develop a more sophisticated parallel tracing platform for large-scale, efficient particle tracing. Second, we would like to investigate the on-the-fly construction of FlowHON. In this manner, the network undergoes dynamic updates during particle tracing, thereby maintaining consistently high performance as the tracing process progresses. This may be useful for unsteady flow fields, assuming the transition patterns do not change abruptly. Third, we would like to seek a better balance between the explainability of our linear transformation-based approach and the power of deep learning. Deep neural networks may be used to identify meaningful higher-order states with even longer dependencies.

ACKNOWLEDGEMENTS

This research was supported in part by the National Natural Science Foundation of China through grant 62372484.

REFERENCES

[1] G. Aldrich, J. D. Hyman, S. Karra, C. W. Gable, N. Makedonska, H. Viswanathan, J. Woodring, and B. Hamann. Analysis and visualization of discrete fracture networks using a flow topology graph. *IEEE Transactions on Visualization and Computer Graphics*, 23(8):1896–1909, 2017.

[2] H. Bhatia, S. Jadhav, P.-T. Bremer, G. Chen, J. A. Levine, L. G. Nonato, and V. Pascucci. Flow visualization with quantified spatial and temporal errors using edge maps. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1383–1396, 2011.

[3] C.-M. Chen, B. Nouanesengsy, T.-K. Lee, and H.-W. Shen. Flow-guided file layout for out-of-core pathline computation. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, pages 109–112, 2012.

[4] C.-M. Chen and H.-W. Shen. Graph-based seed scheduling for out-of-core FTLE and pathline computation. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, pages 15–23, 2013.

[5] C.-M. Chen, L. Xu, T.-K. Lee, and H.-W. Shen. A flow-guided file layout for out-of-core streamline computation. In *Proceedings of IEEE Symposium on Large Data Analysis and Visualization*, pages 115–116, 2011.

[6] G. Chen, Q. Deng, A. Szymczak, R. S. Laramee, and E. Zhang. Morse set classification and hierarchical refinement using conley index. *IEEE transactions on visualization and computer graphics*, 18(5):767–782, 2011.

[7] G. Chen, K. Mischaikow, R. S. Laramee, and E. Zhang. Efficient morse decompositions of vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):848–862, 2008.

[8] L. Chen and I. Fujishiro. Optimizing parallel performance of streamline visualization for large distributed flow datasets. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 87–94, 2008.

[9] F. Chierichetti, R. Kumar, P. Raghavan, and T. Sarlos. Are web users really markovian? In *Proceedings of International Conference on World Wide Web*, pages 609–618, 2012.

[10] D. Edler, L. Bohlin, and M. Rosvall. Mapping higher-order network flows in memory and multilayer networks with infomap. *Algorithms*, 10(4):112, 2017.

[11] A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen, and C. Bischof. Viracocha: An efficient parallelization framework for large-scale CFD post-processing in virtual environments. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, pages 50–50, 2004.

[12] H. Guo, W. He, S. Seo, H.-W. Shen, E. M. Constantinescu, C. Liu, and T. Peterka. Extreme-scale stochastic particle tracing for uncertain unsteady flow visualization and analysis. *IEEE Transactions on Visualization and Computer Graphics*, 25(9):2710–2724, 2018.

[13] H. Guo, J. Zhang, R. Liu, L. Liu, X. Yuan, J. Huang, X. Meng, and J. Pan. Advection-based sparse data management for visualizing unsteady flow. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2555–2564, 2014.

[14] J. Han, J. Tao, and C. Wang. Flownet: A deep learning framework for clustering and selection of streamlines and stream surfaces. *IEEE transactions on visualization and computer graphics*, 26(4):1732–1744, 2018.

[15] T. Höllt, M. Hadwiger, O. Knio, and I. Hoteit. Probability maps for the visualization of assimilation ensemble flow data. In *Workshop on Visualisation in Environmental Sciences*, 2015.

[16] F. Hong, C. Lai, H. Guo, E. Shen, X. Yuan, and S. Li. Flda: Latent dirichlet allocation based unsteady flow analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2545–2554, 2014.

[17] 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)*, pages 76–85, 2018.

[18] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[19] C.-J. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779, 2007.

[20] K. Lu, A. Chaudhuri, T.-Y. Lee, H.-W. Shen, and P. C. Wong. Exploring vector fields with distribution-based streamline analysis. *PacificVis*, 13:257–264, 2013.

[21] J. Ma, C. Wang, and C.-K. Shene. FlowGraph: A compound hierarchical graph for flow field exploration. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 233–240, 2013.

[22] J. Ma, C. Wang, C.-K. Shene, and J. Jiang. A graph-based interface for visualanalytics of 3D streamlines and pathlines. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1127–1140, 2013.

[23] B. Moloney, D. Weiskopf, T. Moeller, and M. Strengert. Scalable sort-first parallel direct volume rendering with dynamic load balancing. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, 2007.

[24] A. Noack. Energy models for graph clustering. *Journal of Graph Algorithms and Applications*, 11(2):453–480, 2007.

[25] B. Nouanesengsy, T. Lee, K. Lu, H. Shen, and T. Peterka. Parallel particle advection and ftle computation for time-varying flow fields. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2012.

[26] B. Nouanesengsy, T.-K. Lee, and H.-W. Shen. Load-balanced parallel streamline generation on large scale vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1785–1794, 2011.

[27] S. Oeltze, D. J. Lehmann, A. Kuhn, G. Janiga, H. Theisel, and B. Preim. Blood flow clustering and applications invirtual stenting of intracranial aneurysms. *IEEE transactions on visualization and computer graphics*, 20(5):686–701, 2014.

[28] T. Peterka, R. Ross, B. Nouanesengsy, T. Lee, H. Shen, W. Kendall, and J. Huang. A study of parallel particle tracing for steady-state and time-varying flow fields. In *Proceedings of IEEE International Parallel Distributed Processing Symposium*, pages 580–591, 2011.

[29] W. Reich and G. Scheuermann. Analysis of streamline separation at infinity using time-discrete markov chains. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2140–2148, 2012.

[30] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[31] M. Rosvall, A. V. Esquivel, A. Lancichinetti, J. D. West, and R. Lambiotte. Memory in network flows and its effects on spreading dynamics and community detection. *Nature Communications*, 5(1):1–13, 2014.
[32] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.
[33] T. Takaguchi, M. Nakamura, N. Sato, K. Yano, and N. Masuda. Predictability of conversation partners. *Physical Review X*, 1(1):011008, 2011.
[34] J. Tao, J. Ma, C. Wang, and C.-K. Shene. A unified approach to streamline selection and viewpoint selection for 3d flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):393–406, 2012.
[35] J. Tao, C. Wang, N. V. Chawla, L. Shi, and S. H. Kim. Semantic flow graph: A framework for discovering object relationships in flow fields. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3200–3213, 2017.
[36] J. Tao, J. Xu, C. Wang, and N. V. Chawla. HoNVis: Visualizing and exploring higher-order networks. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 1–10, 2017.
[37] C. Wang and J. Tao. Graphs in scientific visualization: A survey. *Computer Graphics Forum*, 36(1):263–287, 2017.
[38] J. Xu, T. L. Wickramarathne, and N. V. Chawla. Representing higher-order dependencies in networks. *Science Advances*, 2(5):e1600028, 2016.
[39] L. Xu and H.-W. Shen. Flow web: A graph based user interface for 3D flow field exploration. In *Proceedings of IS&T/SPIE Conference on Visualization and Data Analysis*, volume 7530, page 75300F, 2010.
[40] H. Yu, C. Wang, and K. Ma. Parallel hierarchical visualization of large time-varying 3D vector fields. In *Proceedings of ACM/IEEE Conference on Supercomputing*, pages 1–12, 2007.
[41] H. Yu, C. Wang, C.-K. Shene, and J. H. Chen. Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics*, 18(8):1353–1367, 2011.
[42] J. Zhang, H. Guo, F. Hong, X. Yuan, and T. Peterka. Dynamic load balancing based on constrained K-D tree decomposition for parallel particle tracing. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):954–963, 2018.
[43] J. Zhang, H. Guo, and X. Yuan. Efficient unsteady flow visualization with high-order access dependencies. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 80–87, 2016.
[44] J. Zhang, H. Guo, X. Yuan, and T. Peterka. Dynamic data repartitioning for load-balanced parallel particle tracing. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 86–95, 2018.
[45] J. Zhang and X. Yuan. A survey of parallel particle tracing algorithms in flow visualization. *Journal of Visualization*, 21(3):351–368, 2018.

Nan Chen is currently a Ph.D. student at the Johns Hopkins University. He received a B.E. degree in computer engineering from Sun Yat-sen University in 2020. His research interests are scientific visualization, multivariate data exploration, and graph machine learning.

Zhihong Li is a master student at Sun Yat-sen University. He received a B.E. degree in computer science from Liaoning University in 2020. His research interests are flow visualization and deep learning for scientific visualization.

Jun Tao is an associate professor of computer science at Sun Yat-sen University and National Supercomputer Center in Guangzhou. He received a Ph.D. degree in computer science from Michigan Technological University in 2015. Dr. Tao's major research interest is scientific visualization, especially on applying information theory, optimization techniques, and deep learning to flow visualization and multivariate data exploration.

APPENDIX A
DISCUSSIONS ON EXISTING HIGHER-ORDER NETWORKS

Conventional network models are usually based on the assumption of Markovian behavior, meaning that the transition probability on a specific node only depends on its current state. Recent work has empirically demonstrated that this assumption is insufficient to model the real-world transitions [9], [32], [33]. Rosvall et al. [31] proposed a second-order Markov model composed of memory nodes that encode the currently visited node as well as a previously visited node. Xu et al. [38] put forward a higher-order network (HON) with nodes of variable orders. This approach only generates necessary higher-order nodes which behave significantly differently from their respective lower-order nodes. Edler et al. [10] abstracted different forms of higher-order networks as a sparse memory network that distinguished physical nodes from state nodes encoding higher-order dependencies, and employed a generalized map equation algorithm on it to detect overlapping module patterns. The higher-order network is also used in visualization approaches to examine long-term dependencies. Tao et al. [36] developed a visual analytic framework of HON that allows users to examine higher-order Markov dependencies interactively at different levels of granularity.

These higher-order networks are generally categorized into two types: the fixed-order [31] and variable-order networks [38]. Figure 1 (a) shows a portion of a fixed-order network. In this network, each node is a third-order state, and each edge represents the transition probability between two corresponding states. Transition probabilities are initially recorded between a higher-order state and an individual event, and later expanded into transitions between two states of the same order. Note that the higher-order state provides enough previous history to expand an individual event to a state of the same order or one order higher. For example, the transition $A|B.C \rightarrow F$ is expanded into $A|B.C \rightarrow F|A.B$. The fixed-order network provides complete dependency information up to a fixed order, but its size may increase exponentially with the order.

Figure 1 (b) illustrates a variable-order network, which contains nodes of various orders to reduce the unnecessary higher-order nodes. A higher-order node is included in the variable-order network only if its transition behavior differs from the corresponding lower-order node. The behavior difference is measured between the transition probability distribution of a higher-order node and that of the corresponding lower-order node. The Kullback-Leibler divergence (KLD) [18] is used to calculate the difference between two distributions based on information theory. For example, in Figure 1 (b), the two third-order nodes $A|B.D$ and $A|B.E$ have similar distributions to a second-order node $A|B$, while the third-order node $A|B.C$ has a different distribution. In this case, $A|B.D$ and $A|B.E$ are considered to be redundant, as they provide no additional information than $A|B$. Therefore, the variable-order network will include $A|B$ to represent both $A|B.D$ and $A|B.E$, and include $A|B.C$ to preserve its unique transition behavior.

However, the variable-order network reduces the size from the fixed-order network based on handcrafted rules, which does not guarantee optimal performance for three reasons. First, the construction process only considers the similarities between states of different orders but does not take into account the similarities between states of the same order. As a result, it fails to combine similar states corresponding to different lower-order nodes. Second, the transition probability distribution of the lower-order state may not be the most appropriate one to represent all states reduced to it. For example, in Figure 1, $A|B$ represents $A|B.D$ and $A|B.E$, but its distribution includes the transitions through $A|B.C$ as well, which may lead to inaccurate probabilities. Third, both the variable-order and fixed-order networks estimate the transition probabilities by counting transitions over the entire history, but ignore the differences residing in the transition patterns due to the change of particles' spatial distribution.

To deal with the above mentioned problems, we propose a three-layer construction pipleline to build FlowHON. Our three-layer construction model can be seen as a generalization of both the fixed-order [31] and variable-order [38] networks. The fixed-order network represents each HO-state of a fixed-order as a node, and the probability of starting from a certain node is given by the sampled data. This can be seen as using our approximate assignment (which will be discussed later) to produce the distribution matrix $\mathbf{D}$ and using an identity matrix as the aggregation matrix $\mathbf{A}$. The variable-order network always starts from a first-order state and generates the history in random walks on the network. This can be seen as a distribution matrix $\mathbf{D}$, where $D_{i,j} = 1$ implies that the i-th HO-state is a first-order one. The variable-order network uses HO-states of different orders as nodes. This can be seen as using a handcrafted rule to produce the aggregation matrix $\mathbf{A}$, which aggregates HO-states of higher orders into the corresponding lower-order ones. Both the fixed-order and the variable-order networks estimate the transition probabilities using the sampled data, which can be seen as the computation of $\mathbf{T}$. Therefore, these two network construction algorithms can be special solutions to our problem. Ideally, our approach should produce the optimal performance given the same network size.

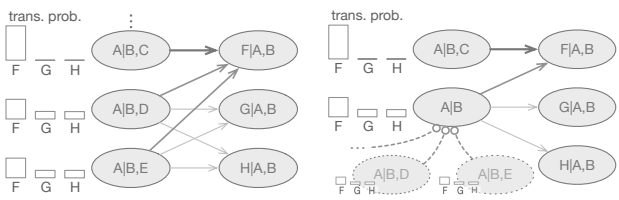


Fig. 1: Existing higher-order network construction algorithms. (a) shows a fixed-order network [31], where all nodes share a fixed order. (b) shows the corresponding variable-order network [38], where similar higher-order nodes are merged into a lower-order node.

APPENDIX B
EXPERIMENT CONFIGURATIONS AND TRAINING SPECIFICS

B.1 Experimental Environment

We conduct the method construction on a server running CentOS Linux 7 (Core), equipped with a 2.10GHz Intel Xeon Gold 6230R processor, 512GB of RAM, and a single NVIDIA A100 Tensor Core GPU featuring 80GB of GPU memory. The software used includes Python 3.8.19, Tensorflow 2.11.0, and CUDA 11.6.

B.2 Network Size and GPU Memory Consumption

To offer practical guidance on selecting appropriate network sizes, we conducted experiments on an unsteady flow dataset to estimate the GPU memory consumption of different approaches under varying time step settings. Specifically, we evaluated the ECMWF dataset with 500 spatial blocks, testing configurations with 10 and 20 time steps. Under the setting of 10 time steps, the generated fixed-order network (Fixed) consists of approximately $18,500$ nodes, requiring around 25 GB of GPU memory for edge optimization. In contrast, the constructed FlowHON consists of approximately $7,200$ nodes, requiring about 14 GB of GPU memory for network construction. When changing the setting to 20 time steps, the size of Fixed grows to approximately $28,800$ nodes, with GPU memory consumption rising to 59 GB for edge optimization. Meanwhile, FlowHON expands to approximately $10,800$ nodes, requiring around 19 GB of GPU memory for network construction. Additionally, we acknowledge that the exact memory consumption and computational efficiency highly depend on hardware configurations and software library versions. For example, in our comparison of implementations using PyTorch and TensorFlow, we observed that TensorFlow sometimes requires less GPU memory but at the cost of higher CPU usage and significantly increased computation time for large networks. We hypothesize that TensorFlow's automatic memory management may offload certain computations to the CPU to conserve GPU memory, albeit at the expense of efficiency.

B.3 Justifications on difference threshold selection

In implementation, we empirically use a difference threshold of 0.04 for the hierarchical clustering based on our observations on the distribution of differences. As shown in Figure 2, the difference value 0.04 is located at the "elbow" of the difference distribution, meaning there is limited room for node compression beyond this point.

B.4 Convergence Analysis of Transition Matrix Training

In this section, we examine the learning dynamics of training the transition matrix. We do this by plotting the loss (as defined in Eq. 5) at each training epoch for the twelve steady flows, as shown in Figure 3. Our analysis reveals that the optimization of the transition matrix reaches convergence by the end of training across all data sets.

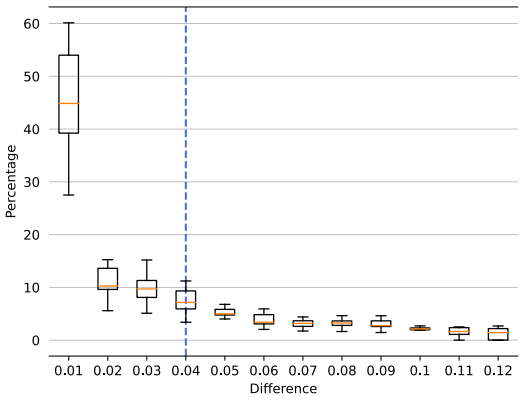


Fig. 2: The difference distribution of all data sets. The x-axis represents the difference range. The Y-axis represents the percentage of difference values in a range. The box plot shows the 25%, 50%, and 75% quantiles of the percentage over data sets. The blue dashed line shows the threshold used in our experiment.

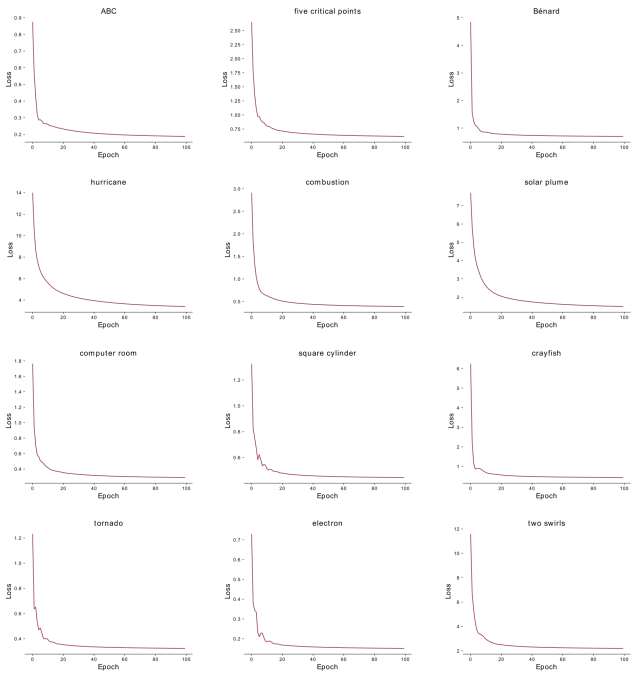


Fig. 3: The loss dynamic (y-axis) over training epochs (x-axis) for each of the twelve steady flows.

B.5 Analysis of the impact of different seeding strategies

Let P denote the distribution of particles where each element $P[i]$ represents the number of particles in block i. Let Q denote the probability of sampling a seed in each block. The original sampling method uniformly samples seeds in the space $Q[0] = Q[1] = \cdots = Q[N-1]$, and traces them forward. This seeding strategy might lead to imbalanced particle distribution among blocks as tracing goes on. To balance the particle distribution, we devise another seed-

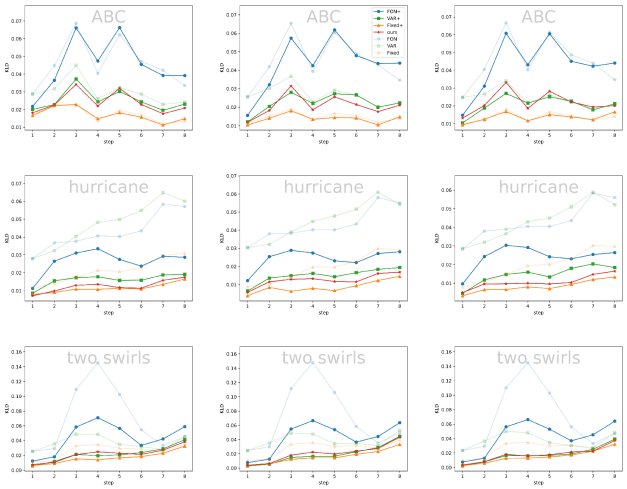


Fig. 4: Experiment result in scenario 1 where the training and testing data are both generated by the new sampling method. Results are from ABC, hurricane, and two swirls data sets. Columns (a) to (c) denote using training sets with 5,000, 10,000, and 20,000 samples.

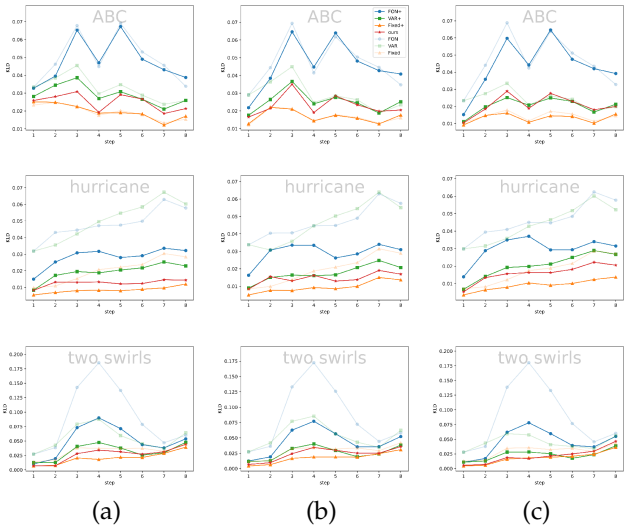
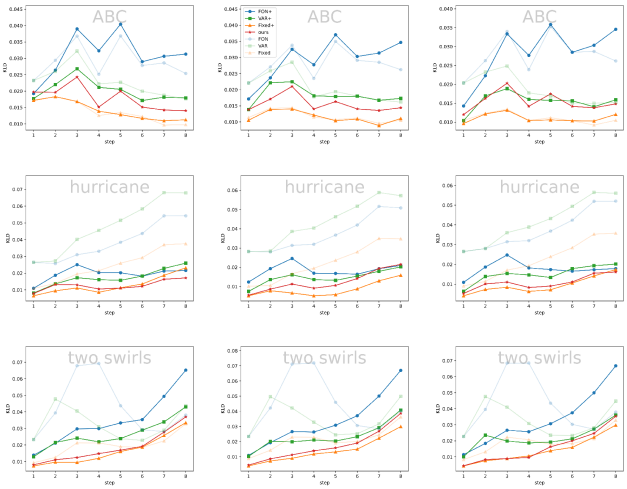


Fig. 5: Experiment result in scenario 2 where the training and testing data are generated by the original sampling method and the new sampling method, respectively. Results are from ABC, hurricane, and two swirls data sets. Columns (a) to (c) denote using training sets with 5,000, 10,000, and 20,000 samples.

ing strategy. This new strategy consists of multiple stages, each of which generates 500 streamlines. Let P_j denote the current particle distribution at the beginning of stage j. In the first stage, the distribution P_0 is assumed to be uniform. During simulation, the probability of sampling a seed in block i is related to the current particle distribution in it:

$$Q_j[i] \propto exp(N - N \cdot P_j[i])$$



od

Fig. 6: Experiment result in scenario 3 where the training and testing data are generated by the new sampling method and the original sampling method, respectively. Results are from ABC, hurricane, and two swirls data sets. Columns (a) to (c) denote using training sets with 5,000, 10,000, and 20,000 samples.

where N is the total number of blocks in the flow field. Overall, this new strategy will iteratively adjust particle distributions to ensure a more balanced distribution. We would like to evaluate the model's performance when using different strategies to generate training and testing data. We consider three scenarios that are described as follows.

- Use the new sampling method to generate the training data as well as test data.
- Use the original sampling method and the new method to generate the training data and test data, respectively.
- Use the new method and the original sampling method to generate the training data and test data, respectively.

For each scenario, we use 15,000 samples for testing, and training data sets are to 5,000, 10,000, or 20,000 samples. The results in these three scenarios are shown in Figure 4, 5, and 6, respectively. These results show that all networks evaluated in the experiment are stable under different seeding strategies for sampling training and testing data. Among the three data sets, FlowHON consistently outperforms FON and VAR, achieving a comparable performance to Fixed+.

B.6 Impact of different block dimensions.

In this subsection, we analyze how different block dimension settings impact the network generation process and the particle density estimation task. For each dataset, we experimented with three grid configurations: the original grid partition used in the main paper, a sparser grid (referred to as the sparse grid), and a denser grid (referred to as the dense grid). For each data set, the sparse grid is configured to have roughly half the number of blocks compared to the

TABLE 1: Comparison of performance across various grid densities, including construction time in seconds, network size based on the number of nodes, and the estimation error in particle density estimation. The result of each method represents an average over 12 steady flow fileds under specific grid settings. Note that for FON, the network size corresponds to the number of blocks in the grid partition.

	construction time (s)			network size			estimation error		
method	sparse	original	dense	sparse	original	dense	sparse	original	dense
FON	0.45	0.69	1.30	63	127	427	0.034	0.042	0.049
VAR	0.67	0.99	1.93	370	693	1842	0.033	0.039	0.048
Fixed	0.30	0.40	1.47	871	1607	4308	0.022	0.028	0.035
FON+	3.87	4.97	10.44	63	127	427	0.021	0.023	0.030
VAR+	7.32	16.58	94.19	370	693	1842	0.012	0.016	0.028
Fixed+	18.01	59.68	508.48	871	1607	4308	0.009	0.013	0.022
ours	20.32	31.05	73.44	324	454	868	0.012	0.016	0.027

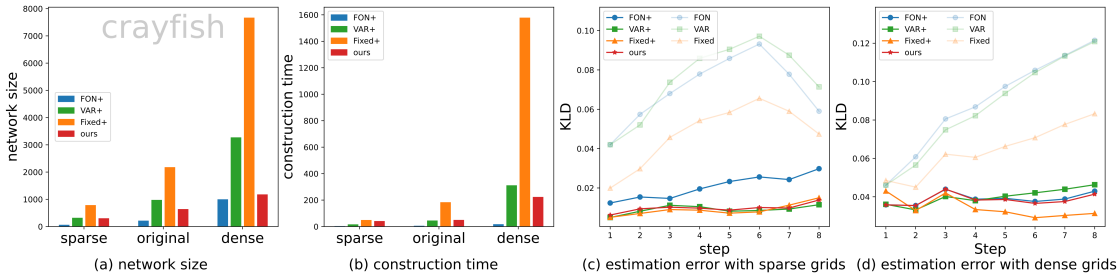


Fig. 7: Comparison of performance across various grid densities in crayfish data set. (a) shows how the network size change in different grid densisties for different methods. (b) shows how the construction time change. (c) and (d) are the estimation error of particle density over block steps under sparse grid and dense grid, respectively.

original, while the dense grid is set to have approximately three times the number of blocks. For each grid configuration, we applied different network construction methods and evaluated their performance in particle density estimation, while keeping all other settings unchanged. The results are summarized in Table 1, with detailed outcomes for the crayfish dataset provided in Figure 7.

In terms of network construction time, we observed that grid density has a more significant impact on methods with edge optimization, particularly Fixed+. This is likely due to the fact that the network size of Fixed increases rapidly as grid density increases, resulting in a larger number of edges that need optimization. For variable-order methods such as VAR+ and FlowHON, the increase in construction time is more moderate, as these methods will group redundant HO-states into HO-nodes to improve efficiency. Notably, at the dense grid, FlowHON achieves a significantly smaller network size compared to VAR, demonstrating the effectiveness of our HO-state grouping strategy. As grid density increases, the flow field becomes partitioned at a finer level, making flow patterns within each block simpler and reinforcing the importance of advanced grouping strategies to reduce redundant HO-states. Regarding the estimation error, we found that the performance gap between FON and higher-order networks decreases as the grid becomes denser. This is expected as denser grids result in simpler flow patterns within each block. Across all block dimension settings, our method consistently achieves a performance closest to Fixed+, while maintaining a much smaller network size.

TABLE 2: Average estimation error across 12 data sets when using different distance metrics to measure distance between HO-states.

distance metrics	estimation error
Euclidean	0.016
KL-Divergence	0.015
Cosine similarity	0.018

B.7 Impact of different distance metrics

In this subsection, we investigate the impact of different distance metrics used to calculate the distance between HO-states. Specifically, we evaluate three metrics: Euclidean distance (used in the main paper), KL-Divergence, and cosine similarity. For each metric, we constructed FlowHON and tested it on the particle density estimation task on 12 steady flow data sets. The results, presented in Table 2, indicate that the overall model performance is not highly sensitive to the choice of distance metric. Among the three, cosine similarity delivers the lowest performance, while KL-Divergence performs slightly better than the Euclidean distance, which was used in our main paper.

APPENDIX C
MORE RESULTS ON STEADY FLOW

C.1 More results on particle density estimation

Figure 8 shows the error estimation over block steps for the other data sets.

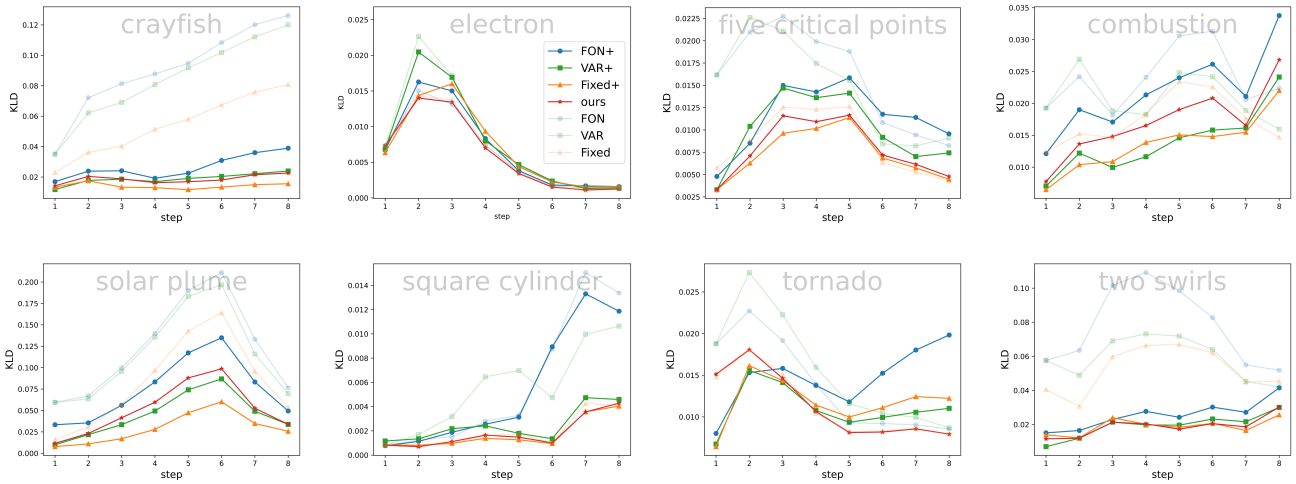


Fig. 8: Estimation error of particle density (y-axis) over block steps (x-axis) for eight data sets. The blue, green, orange, and red curves show the accuracy of FON, VAR, Fixed, and FlowHON, respectively. Curves with lighter colors show the estimation error of respective original networks without our edge optimization, and curves with darker colors show that of optimized networks.

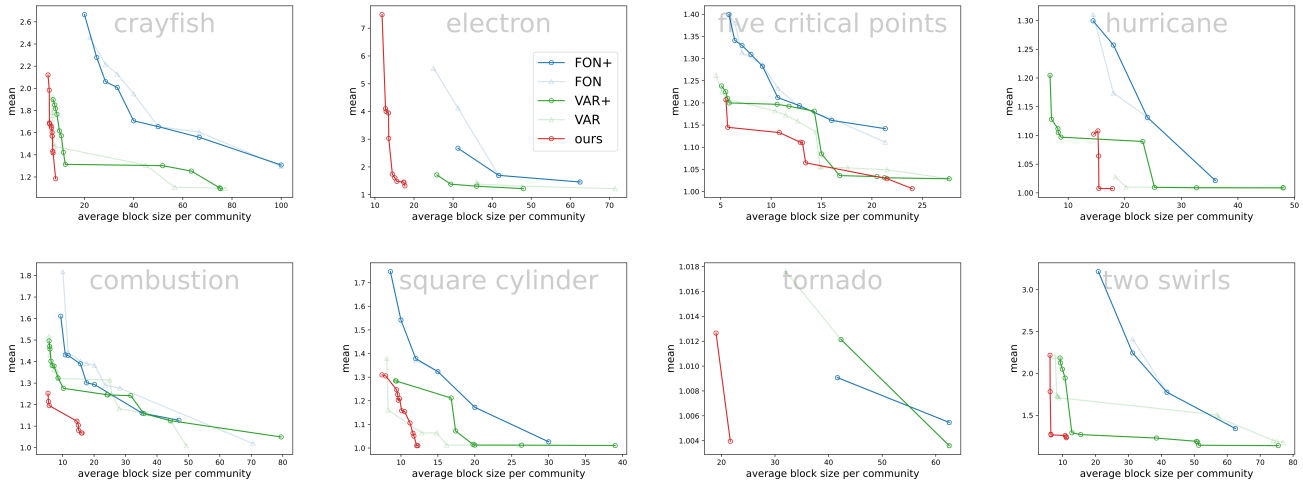


Fig. 9: Community detection results on eight data sets. The x-axis denotes the average community size regarding the number of blocks, and the y-axis denotes the mean of the number of communities visited. For the same community size, smaller means of community visits are preferred. Blue, green, and red curves denote the results of FON, VAR, and FlowHON, respectively. Curves with lighter colors show the result of respective original networks without edge optimization, and curves with darker colors show that of optimized networks.

C.2 More results on community detection

Figure 9 shows the community detection results for the other data sets.

APPENDIX D
VISUAL EXPLORATION ON MORE DATA SETS

Solar Plume. The solar plume data set is partitioned into $4 \times 4 \times 10$ data blocks. In Figure 10 (a), we find that the nodes in FlowHON form eight distinct groups in the graph layout. The four groups at the center (red, orange, gray, and light purple) correspond to the four sectors of the crown of the solar plume. The node groups and the sectors follow the same order (as indicated by the black dashed arrow), preserving the neighboring relations between groups. The node groups at the outer ring (green, purple, brown, and yellow) correspond to the tail of the solar plume. They also follow the same order, and each group stays close to the corresponding group in the inner layer. In Figure 10 (c), FON does not distinguish the four sectors at the crown clearly. Instead, the upper and lower blocks form two groups of nodes (blue and orange) in the graph. We can see two clear paths in the upper region of the graph, corresponding to the blocks in the front. We can also see a path in the bottom-left region, but the fourth path is not clearly visible.

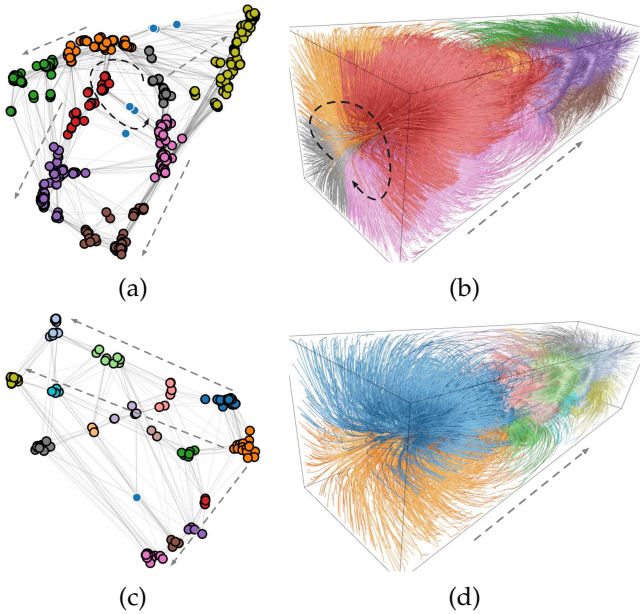


Fig. 10: Solar plume data set: visualization of networks and streamline segments related to selected node groups within the solar plume data. The first row illustrates the network and streamline segments of FlowHON, while the second row depicts those of FON. The black dashed arrows illustrate the order correspondence between node groups and sectors within the flow field, from which we can tell that the layout of FlowHON preserves more meaningful neighboring relations between node groups for visual exploration. The light blue spheres in subfigure (a) represent nodes that have not been selected.

APPENDIX E
MORE RESULTS ON UNSTEADY FLOW

E.1 Visual Exploration with Geo-Reference

Figure 12 is an alternative version of Figure 10 with geo-reference.

E.2 Quantitative Results on 4 Unsteady Flows with More General Partitioning Strategies

In Section 5.5 of our preliminary experiments, we focused on partitioning unsteady flow fields solely along spatial dimensions. Pathlines generated at each timestep were treated as streamlines for subsequent processing. In this part, we conduct further experiments on unsteady flow fields to complement the preliminary experiments. Specifically, we partition the unsteady flow fields along both spatial and temporal dimensions, creating space-time blocks for the network construction. To enhance the comprehensiveness of our evaluation, we include two additional data sets, vsfs9 and VEC. Table 3 provides a summary of the statistics for the unsteady flow fields used, along with details of their partitioning. The construction procedure applied to these unsteady flow fields is identical to that used for steady flows, with space-time blocks being treated analogously to spatial blocks.

Figure 11 demonstrates the performance of different networks in particle density estimation and community detection tasks. The patterns observed align with those discussed in previous sections. For density estimation, our proposed edge optimization enhances performance for both FON and VAR by a large margin. Across the four unsteady flows examined, FlowHON consistently surpasses FON+ and VAR+, showing comparable results to Fixed. In community detection, FlowHON consistently outperforms FON+ and VAR+ across all datasets, characterized by particles visiting fewer communities with reduced community sizes. These findings highlight the adaptability and effectiveness of FlowHON and its edge optimization component in modeling more complicated patterns in unsteady flow fields.

TABLE 3: Statistics of the unsteady flow fields used in the experiment. Space-time block dimensions show the numbers of blocks along each axis, where the last axis represents time.

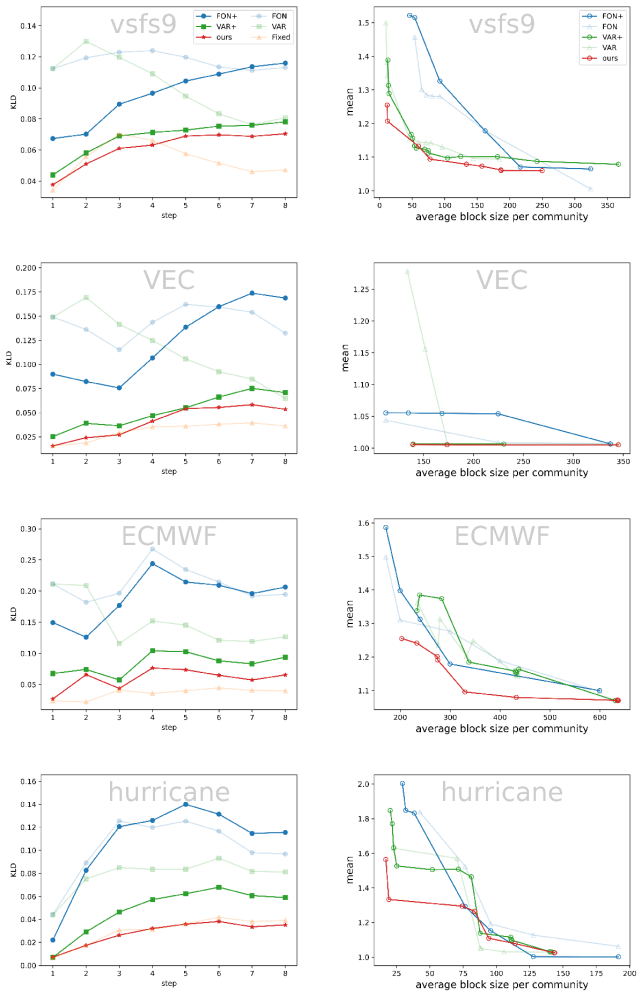
data set	data dimension	space-time block dimension
vsfs9	$40 \times 56 \times 68 \times 36$	$6 \times 8 \times 8 \times 6$
VEC	$147 \times 147 \times 43 \times 120$	$8 \times 8 \times 1 \times 6$
ECMWF	$480 \times 241 \times 15 \times 44$	$15 \times 10 \times 1 \times 12$
hurricane	$500 \times 500 \times 100 \times 48$	$10 \times 10 \times 1 \times 4$

APPENDIX F
DOMAIN EXPERT FEEDBACK

We invite Dr. X, an expert in fluid dynamics, to evaluate the effectiveness of FlowHON. Dr. X has more than ten years of experience in this field, and her recent work focuses on developing scalable computational tools for various scientific domains, including atmospheric science. The evaluation was performed in two stages. In the first stage, Dr. X was introduced to the basic concepts of FlowHON and the exploration interface using the tornado data set. In the second stage, Dr. X used the interface to explore the ECMWF data set and compared the FON and FlowHON on it, as this data set is closely related to his application of interest.

Dr. X stated that "FlowHON is useful in discovering features as the complex flow regions often seem to be denser than other regions." This may be related to the fact that FlowHON will produce more HO-nodes in the blocks where particles exhibit more diverse transition patterns. He further commented that "The higher-order nodes separate the feature patterns from the other flow lines, which is useful to study the specific physical phenomenon. For example, in the atmospherical data sets, the spiraling pattern, such as typhoons, is often studied. FlowHON enables the selection of this kind of features, but in FON, this pattern is hidden in the laminar flows and cannot be selected." This is indeed consistent with the findings in our exploration. However, in terms of the overall structure of the graphs, Dr. X stated that "The graph visualization of the two networks (FON and FlowHON) reveal similar structures."

Dr. X also mentioned some desired features that may be supported by FlowHON in the future. She stated that "When an HO-node is selected, it will be more convenient if some functions are provided to explore its neighbors in the graph. For the atmospherical science, this may be helpful to study the water vapor transmission, the pollution diffusion, and the energy circulation." As FlowHON tends to provide a more deterministic transition behavior among



(a) (b)

Fig. 11: The particle density estimation (a) and the community detection (b) results for the unsteady flow fields. The unsteady flow fields are partitioned along spatial-temporal dimensions. The x-axis in column (a) represents the space-time block step, while that in column (b) represents the average community size defined by the number of space-time blocks. The y-axis in column (a) represents the estimation error of particle density, while that in column (b) represents the average number of communities visited. The first row to the fourth row illustrates the result of the vsfs9, VEC, ECMWF, and hurricane data sets, respectively. Note that we do not apply edge optimization to Fixed as the GPU memory consumption exceeds our GPU's memory capacity.

nodes, we feel that this is a promising direction to explore. For example, starting from one node, FlowHON can better estimate the regions that will be influenced by the selected node. She also mentioned that "FlowHON is an interesting idea to handle the dependencies in computation. It will be interesting to see whether FlowHON can work with the computation tools we are developing."

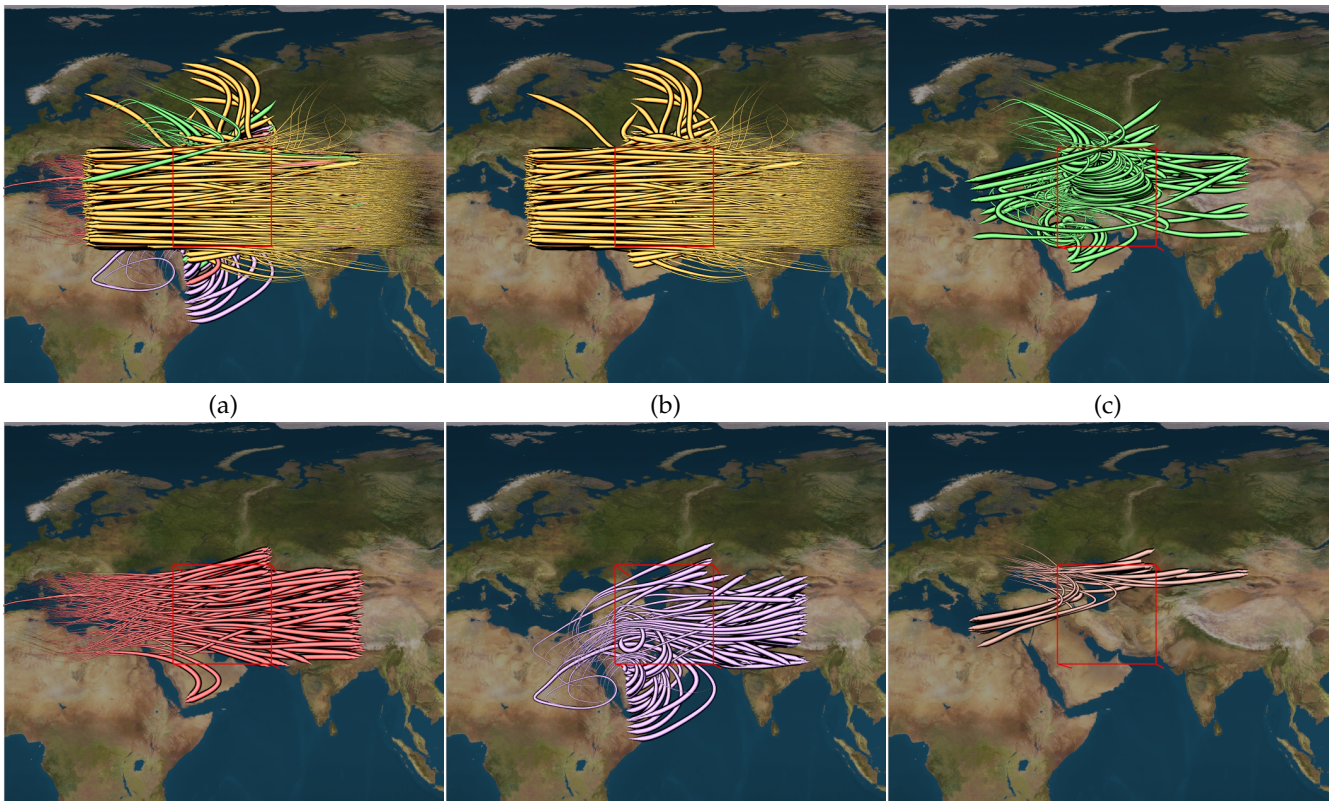


Fig. 12: Flow patterns related to a single block in the unsteady ECMWF data set. (a) to (d) show the pathlines related to the four major HO-nodes in the block highlighted in red box, respectively. Red arrows in (a) and (b) point out the direction of pathlines. (e) and (f) shows the pathlines of different HO-nodes at the time step 2 and 8, respectively. The HO-nodes are produced by our FlowHON approach.