TP-Bundle: Interactive Hierarchical Edge Bundling for Large Graphs with Transformer-based Prefetching

Xinyan Xie, Jianxin Sun, Claire X. Shen, Hongfeng Yu
University of Nebraska-Lincoln, Lincoln, NE, USA

***Abstract*—Edge bundling is a widely used technique for graph visualization, offering advantages such as reducing visual clutter, revealing macroscopic connectivity patterns, and supporting multiscale exploration. However, applying edge bundling to large-scale graphs remains challenging due to the high computational cost of computing edge trajectories and the significant I/O overhead of managing massive datasets. Existing acceleration methods improve performance but are typically limited to small graph datasets. In this paper, we present *TP-Bundle*, an interactive hierarchical edge bundling framework designed for large graphs with Transformer-based prefetching. The framework integrates three components. First, we construct a hierarchical tree structure via community detection and define a three-dimensional exploratory space to represent user navigation. Second, we design a Transformer-based neural network with attention to predict the most likely next view, guiding a predictive prefetching mechanism. This reduces I/O latency by proactively loading data into memory during out-of-core visualization. Finally, we combine these components into a unified pipeline that enables interactive edge bundling on massive graphs. Experimental results demonstrate that our TP-Bundle framework consistently reduces rendering latency while maintaining high visual quality, enabling scalable and responsive graph exploration at unprecedented scales. To our knowledge, this is the first framework to combine hierarchical traversal with deep learning–based prefetching for interactive edge bundling at scale.**

***Index Terms*—Graph visualization, edge bundling, prefetching**

I. Introduction

Graphs are abstract structures composed of nodes (vertices) and edges that capture relationships between entities. They provide a flexible framework for representing diverse systems such as social networks, biological interactions, communication systems, and transportation infrastructures. In many applications, nodes and edges are associated with attributes such as locations, weights, directions, or labels, which enrich the representation and enable more detailed analysis. By combining structural information with contextual data, graphs support effective modeling, analysis, and visualization.

Visualizing large and complex graphs is a challenging task. A straightforward approach is to draw node-link diagrams, where nodes are placed in space and edges are drawn as straight or curved lines. While effective for small datasets, this method quickly becomes unreadable at scale. Even graphs with thousands or millions of edges can cause the visualization to degenerate into severe visual clutter, often referred to as the hairball problem. In real-world scenarios, graphs often contain far more edges, making the challenge even more significant. To address this issue, edge bundling algorithms [19], [60] have been developed. Edge bundling reduces clutter by grouping similar edges so that they follow shared, smooth paths, creating the visual effect of cables or flows. This approach highlights dominant structures and relationships, making large graphs more interpretable while suppressing distracting visual noise.

However, edge bundling on large-scale graphs faces two primary challenges. The first is high computational complexity. Unlike simply drawing straight edges, bundling dynamically reshapes them so that similar edges converge along shared paths. This requires algorithms to calculate similarities among edges based on spatial proximity, direction, or shared endpoints. For large graphs, these comparisons can approach quadratic complexity if not mitigated with spatial indexing structures such as grids, quadtrees, or k-d trees. Iterative optimization further increases the cost, as methods like force-directed bundling [60] repeatedly update edge segments through simulated attraction and repulsion forces, requiring many rounds of position updates and relationship recalculations.

The second challenge is the I/O bottleneck. Edge bundling involves storing, transferring, and rendering a large volume of graph data, and moving this data across the memory and storage hierarchy can significantly slow down the visualization process. The problem becomes even more severe in interactive settings, where users expect smooth panning, zooming, and rotation, often necessitating the recomputation or incremental updates of bundles in real-time. Such intensive data movement places heavy demands on the memory and I/O hierarchy, which limits performance and responsiveness.

Several approaches have been proposed to accelerate edge bundling through distributed computing [40] and GPU-based methods [45]. However, scaling edge bundling is still challenging due to the increasing computational and I/O costs associated with ever-growing graph data sizes. Our method is motivated by two key observations. First, the screen space available for visualization is limited, which means it is often unnecessary to load and render the entire dataset at once. Second, user exploration in graph visualization typically follows a global-to-local pattern. Users often begin with a global overview, then zoom into a localized region of interest, and iteratively continue this process by moving from one region to another rather than jumping arbitrarily across the graph. These observations motivate a hierarchical visualization strategy that adapts the level of detail to the current view while alleviating both computational and I/O overhead. Based on this idea, we introduce an interactive visualization method tailored for large-scale graphs. Unlike existing approaches that assume the entire dataset must be processed at once or rely solely on

979-8-3315-9447-3/25/$31.00 ©2025 IEEE

low-level acceleration, our method adapts to both the structural properties of large graphs and the behavioral patterns of user exploration. Our pipeline integrates hierarchical community detection, constructs an exploratory space, and employs a Transformer-based deep learning prefetching algorithm, all aimed at reducing average rendering latency and enhancing interactive performance. The main contributions of this work are as follows:

- A visualization method based on hierarchical traversal of the graph structure rather than direct rendering of the entire dataset.
- Construction of a three-dimensional exploratory space that incorporates both the spatial location of graph nodes and the hierarchical depth of the graph tree.
- A Transformer-based deep learning prefetching policy that predicts the most likely next view, enabling parallel data loading and rendering.

Based on these components, we refer to our overall approach as *TP-Bundle*, a *Transformer-based Prefetching Edge Bundling* framework designed to support interactive hierarchical visualization of large graphs. The remainder of this paper is organized as follows. Section II reviews related work, Section III presents the architecture of our method, Section IV reports experimental results and evaluation, and Section V concludes the paper.

II. Related Work

A. Edge Bundling

Edge bundling has emerged as a central technique to reduce visual clutter in node-link diagrams by aggregating edges with similar spatial or structural characteristics. Early geometry-based approaches emphasized spatial embedding, such as Geometry-based Edge Bundling (GBEB) [6] and Force-Directed Edge Bundling (FDEB) [19], which modeled edges as flexible polylines attracted by simulated forces. Related methods, including Confluent Drawing [9], Divided Edge Bundling (DEB) [59], and Mesh-based Edge Bundling (MBEB) [35], extended this paradigm by merging edges into shared tracks, routing through controlled regions, or exploiting mesh structures. Together, these methods established edge bundling as an effective means of revealing global patterns in dense networks.

Image- and pixel-based techniques reframed bundling as an image-processing problem. Kernel Density Estimation Edge Bundling (KDEEB) [20], Skeleton-based Edge Bundling (SBEB) [10], Multi-Level Density Bundling (MLEB) [43] and Moving Least Squares Edge Bundling (MLSEB) [47] iteratively shifted edge points toward density ridges or extracted skeletons from density fields. Other raster-based methods guided bundles using gradients [14] or anisotropy [24].

Hierarchy- and application-driven extensions further broadened the scope of bundling. Hierarchical Edge Bundling (HEB) [16] and its radial [18], 3D [16], and nested [36] variants exploited tree structures to organize edges. Specialized approaches, such as Winding Roads [23], Flow-based Bundling [13], and Stress-Minimization Bundling [1], introduced semantic control, physical flow dynamics, or optimization criteria. Recent work has targeted temporal networks [44] and multilayer networks [31], ensuring coherence in dynamic or multi-relational contexts. Collectively, these advances established edge bundling as a foundational component of modern graph visualization.

B. Large-scale Data Visualization

The advent of big data has fundamentally transformed the requirements for edge-bundling techniques, as methods originally designed for graphs with tens of thousands of nodes and edges fail to scale to datasets containing millions or more [19], [25]. These limitations manifest in the quadratic growth of pairwise compatibility computations [19], memory constraints, and the breakdown of interactive response times, which are essential for exploratory analysis. This has motivated shifts from pairwise forces to image-space density methods [20], GPU-accelerated pipelines [41], and FFT-based acceleration [26]. The demand for large-scale graph visualization spans diverse domains, including social networks, software dependency graphs, transportation systems, and scientific datasets requiring real-time visualization of massive connectivity patterns [15], [17], [29], [57].

Contemporary large-scale bundling research has converged on three complementary performance strategies. CPU-based multilevel coarsening, exemplified by MINGLE, employs hierarchical graph coarsening to reduce computational complexity while minimizing visual clutter on standard hardware [12]. GPU-based approaches, such as CUBu, relocate the entire bundling pipeline—density estimation, gradient advection, smoothing, and rendering—to graphics hardware, achieving interactive rates on graphs with about 10^6 edges and nearly $50\times$ speedups over prior methods [41]. FFT-based acceleration techniques, such as FFTEB, transform kernel-density computations to the frequency domain, enabling efficient processing of very large graphs while preserving visual quality [26].

Beyond performance optimization, large-scale bundling systems emphasize expressiveness and analytical control. Attribute-aware models preserve direction, temporal information, and weights by separating flows within or across bundles [34]. GPU frameworks incorporate level-of-detail rendering, directional separation, and multi-scale analysis to support exploration at multiple granularities [41]. Recent developments include three-dimensional and immersive bundling for volumetric representations [61], domain-specific applications such as voxel-based or connectome visualization [62], and scalable deployment on the web via WebGL [45], [46]. These advances have broadened the adoption of edge bundling across mobility and transportation analysis, communication and infrastructure networks, biological pathways and connectome visualization, and behavioral studies such as eye-tracking and user interactions. At the same time, surveys and evaluations highlight ongoing challenges in balancing efficiency with visual clarity, developing benchmarks, and establishing principled approaches for attribute-aware bundling at scale [25], [29], [48].

Existing edge bundling methods focus on efficiency through optimizations, GPU acceleration, or frequency-domain processing, as well as expressiveness with attribute-aware or application-specific extensions. However, they typically assume the entire dataset must be processed, which limits interactivity for very large graphs. Our method introduces a hierarchical visualization strategy with a three-dimensional exploratory space and a Transformer-based deep learning prefetching algorithm, reducing computation and I/O overhead while enabling smooth interactive performance. To our knowledge, it is the first bundling framework to combine hierarchical traversal with deep learning–based prefetching for large-scale graph visualization.

C. Caching and Prefetching

Out-of-core visualization systems were developed to overcome memory bottlenecks in large-scale data visualization by dynamically loading only the relevant data subsets during interaction [5], [33], [37]. Central to their efficiency are caching strategies that exploit temporal and spatial locality [4], [7]. Early work borrowed classical cache policies such as LRU and LFU [4], later extended for visualization workloads. Domain-specific strategies were introduced for volume rendering [30] and terrain visualization [28], while adaptive techniques further improved responsiveness in hierarchical and multi-resolution contexts [38]. These methods established caching as a foundational tool to reduce I/O overhead in interactive rendering.

Prefetching complements caching by proactively retrieving data based on predicted user navigation. Early approaches focused on sequential and view-dependent prediction in scientific visualization [5], while later methods adopted statistical models such as Markov chains [3], [8]. Domain-specific extensions incorporated semantic relevance and importance-driven strategies [42], [50], enabling prioritization of application-critical data. However, the unpredictability of exploratory user behavior remained a core challenge for handcrafted prefetching heuristics.

More recently, machine learning techniques have been applied to caching and prefetching in visualization systems. Learned index structures [22], reinforcement learning for cache replacement [58], and neural sequence models for user interaction prediction [39] demonstrate the potential of adaptive, data-driven approaches. These methods leverage deep architectures [2] and transfer learning [32] to generalize across datasets and user populations, while federated learning has been proposed to address privacy concerns in distributed environments [21]. Collectively, this evolution from classical caching policies to machine learning–driven prefetching highlights the trajectory toward autonomous, adaptive visualization systems that can optimize themselves through continuous user feedback.

In contrast, our approach is tailored to graph visualization. By combining a hierarchical graph representation with a Transformer-based predictive model, we leverage the structured, continuous nature of user exploration to guide prefetching. This integration enables more accurate anticipation of future views and reduces I/O latency in interactive edge bundling for large-scale graphs.

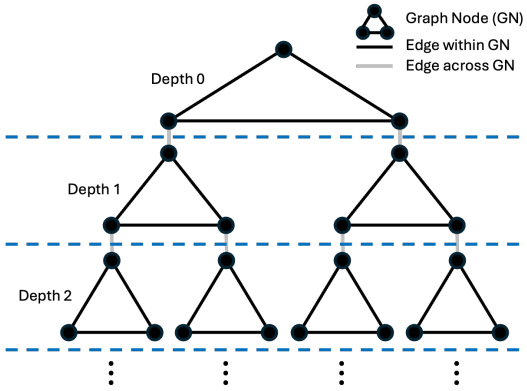


Fig. 1: Hierarchical tree representation of the large-scale graph. Each node in the tree corresponds to a detected community Graph Node (GN), while edges between nodes represent inter-community connections across hierarchical levels. This structure organizes the graph into multiple scales, enabling navigation and visualization at different levels of detail.

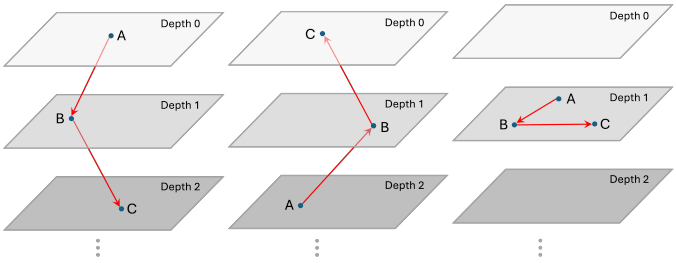
III. Method

To enable interactive edge bundling on large-scale graphs, we design a visualization framework, named TP-Bundle, that reduces computational and I/O overhead while preserving visual fidelity. Instead of processing the entire dataset at once, our method leverages a hierarchical representation of the graph, models user exploration patterns through an exploratory space, and predicts future navigation with a Transformer-based predictive neural network. By integrating this predictive model with a prefetching and caching mechanism, the system overlaps data loading with rendering, thereby minimizing latency and ensuring smooth interactivity. The following subsections describe each component of the pipeline in detail.

A. Community Detection

Visualizing a large-scale graph as a single edge bundling result is often neither effective nor practical. The limited screen space, compared with ever-growing graph sizes, leads to severe clutter and occlusion that obscure meaningful patterns. At the same time, the computational cost of processing all nodes and edges simultaneously is prohibitive, making interactive exploration infeasible. To overcome these challenges, we adopt a hierarchical visualization strategy that renders only subsets of the graph at a time, while allowing users to interactively navigate across different regions as needed. This approach requires preprocessing the raw graph to construct a hierarchical structure that organizes the data for scalable visualization.

In this work, we achieve this preprocessing through community detection methods [11], [27], [52]–[56]. Community detection identifies groups of nodes, referred to as communities or clusters, that are more densely connected to each other than to the rest of the graph. By recursively partitioning



(a) Downward (b) Upward (c) Lateral

Fig. 2: Examples of traversal patterns across Graph Nodes at different depths of the Graph Tree. Three user exploration modes are illustrated: (a) downward traversal from a parent to its child community, (b) upward traversal from a child to its parent, and (c) lateral traversal across sibling communities. In each case, the exploratory trajectory begins at node A and ends at node C, showing how user navigation can be represented as trajectories in the exploratory space.

the graph into communities, we can generate a multilevel representation, or graph hierarchy, that reflects the underlying modular structure of the network. Each level of the hierarchy aggregates fine-grained details into progressively coarser groups, enabling visualization at multiple resolutions. This hierarchical representation not only reduces visual clutter but also preserves important structural information, such as inter-community relationships and cut edges. It reveals hidden patterns and subnetworks that are difficult to discern in a flat representation of the graph. Moreover, it provides the foundation for our exploratory visualization framework, where user navigation is naturally expressed as traversal across different levels of the hierarchy.

B. Exploratory Space

To better capture user exploration behavior, we introduce the concept of an Exploratory Space (ES). After applying community detection, the large-scale graph is summarized into a hierarchical tree structure, which we refer to as the Graph Tree (GT). Each node in the GT corresponds to a detected community, which we call a Graph Node (GN). An example of such a tree with depth three is illustrated in Figure 1. Within each GN, edges connect nodes at the same level, while edges between GNs represent inter-community or cut edges. The Graph Tree is now a union of all Graph Nodes located at various tree depths:

$$GT = \bigcup_{k=0}^{p-1} GN_k \tag{1}$$

where p is the total number of Graph Nodes detected from a community detection method.

We define the ES as a three-dimensional space constructed from two components: (1) the 2D coordinates of each GN in the bundled graph layout (its position in 2D edge bundling visualization), and (2) its depth in the GT hierarchy. Therefore,

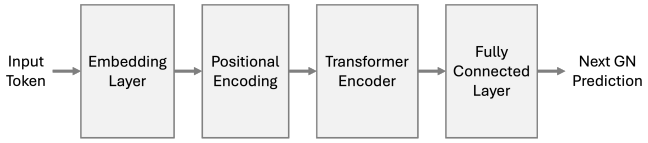


Fig. 3: Architecture of the proposed predictive neural network. The model consists of an embedding layer, positional encoding, a Transformer encoder with multi-head attention, and a fully connected output layer for predicting the next GN.

a user's interaction with the visualization can be represented as a trajectory through this ES, where movements reflect both spatial navigation within the 2D view and transitions across hierarchical levels. A specific user exploratory trajectory (T), consisting of m sequentially visited GNs, can be described as:

$$T = \{GN_{x_0,y_0,d_0}, GN_{x_1,y_1,d_1}, ..., GN_{x_{m-1},y_{m-1},d_{m-1}}\} \tag{2}$$

where x and y are the GN's coordinates in the 2D bundled layout, and d denotes its depth in the GT. This representation captures navigation not only within a single level but also across multiple depths of the hierarchy. Figure 2 illustrates typical traversal patterns, including drilling down into deeper levels, moving upward toward coarser clusters, and shifting laterally among communities at the same depth. By embedding exploration paths in the ES, we can explicitly model and anticipate user movement, forming the basis of our predictive prefetching mechanism.

C. Traversal Predictive Neural Network

To anticipate user navigation across the graph hierarchy, we design a predictive neural network based on the Transformer encoder architecture. The choice of a Transformer is motivated by three considerations. First, the input to the model is a sequence of GNs with variable lengths depending on the depth of the current view in the GT. Unlike recurrent architectures such as LSTMs [51], which require fixed-length sequences, Transformers can naturally process sequences of arbitrary length. Second, Transformers with attention are highly parallelizable and can process entire sequences simultaneously, achieving better GPU utilization and faster training than sequential RNNs. Third, the attention mechanism provides expressive contextual representations by computing weighted combinations of all tokens, which allows the model to capture richer structural dependencies. The overall architecture of the predictive model is illustrated in Figure 3. It consists of four key stages: an embedding layer, positional encoding, a Transformer encoder, and a fully connected output layer.

1) Embedding Layer: The input to the network is the sequence of previously visited GNs leading to the current GN. Depending on the position of the current GN within the GT, the input sequence can vary in length from 1 to $n-1$, where n is the depth of the GT. The embedding layer maps GN indices into dense vectors. The embedding dimension is a fixed hyperparameter chosen to balance representational capacity and computational cost. Embeddings are necessary because

Algorithm 1 Training the Traversal Predictive Neural Network

Input: Training trajectories T, Graph Tree depth n
Output: Trained Transformer-based predictive model

for each trajectory t in T **do**
  Segment t into sequences $S = \{s_1, s_2, \dots\}$ according to GT depth n
  for each sequence s in S **do**
    Convert GN indices in s into embeddings
    Add positional encodings to embeddings
    Pass through Transformer encoder layers
    Apply fully connected layer to obtain logits
    Compute prediction loss using MSE between logits and ground truth
    Backpropagate error and update model parameters
  end for
end for
Return trained predictive model

neural networks cannot directly operate on raw token IDs; they provide a continuous representation that captures similarity relationships between GNs.

2) Positional Encoding: Since Transformers lack an inherent notion of order, positional encodings are added to the embeddings to inject sequence information. We employ sinusoidal positional encoding, defined as:

$$PE_{pos,2i} = \sin\left(\frac{pos}{1000^{2i/d_{model}}}\right) \tag{3}$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{1000^{2i/d_{model}}}\right) \tag{4}$$

where pos is the position in the sequence and i is the dimension index. Even dimensions use sine, while odd dimensions use cosine. This encoding provides a smooth and continuous representation of position, allowing the model to infer relative order from mathematically related encodings.

3) Transformer Encoder: The Transformer encoder applies several components in sequence:

- **Multi-Head Self-Attention (MHSA):** computes attention weights across tokens, enabling the model to attend to multiple aspects of the sequence simultaneously.
- **Residual Connections:** add the input back to the output of each sublayer, improving gradient flow and enabling the training of deeper networks.
- **Layer Normalization:** standardizes feature dimensions for each sample, stabilizing network dynamics by maintaining consistent mean and variance.
- **Feed-Forward Network (FFN):** applies two fully connected layers with ReLU activation. While self-attention mixes information across positions, the FFN performs per-token computation on the aggregated context.

4) Fully Connected Layer: The final fully connected layer projects the Transformer's hidden representation back to the GN vocabulary space. The input is the hidden state at the last time step, and the output is a probability distribution over candidate GNs.

Algorithm 2 Interactive Hierarchical Edge Bundling of Large-Scale Graphs with Prefetching

Input: Sequence of *GNs*
Output: Edge bundling image of the next *GN* selected by the user

$edge\_bundling\_done \leftarrow false$
// Retrieve current *GN*
Find all files (node and edge files) for current *GN* as $files\_GN$
// Cache management
for each $file$ in $files\_GN$ **do**
  if $file$ in cache **then** // Hit
    Continue
  else// Miss
    if Cache not full **then**
      Load $file$ to cache
    else
      Replace least recently used (LRU) data in cache with $file$
    end if
  end if
end for
// Edge bundling rendering
Compute visualization image from $files\_GN$
if Rendering finished **then**
  $edge\_bundling\_done \leftarrow true$
  Return rendered image
end if
// Prefetching in parallel with rendering
Predict the next GN_{next}
Fetch required files for GN_{next} into $files\_GN_{next}$
for each $file$ in $files\_GN_{next}$ **do**
  if $file$ not in cache **then**
    if $edge\_bundling\_done == false$ **then** // Prefetch
      if Cache not full **then**
        Load $file$ to cache
      else
        Replace LRU data in cache with $file$
      end if
    else
      Break // Early stop prefetching
    end if
  end if
end for

5) Training Procedure: The predictive network is trained on user trajectory data collected during simulated exploration of test graphs. These test graphs are generated by randomly constructing GNs and their connections. Each exploration trajectory is segmented into sequences of varying lengths depending on the GT depth, forming the training dataset. The model is optimized using Mean Squared Error (MSE) loss between the predicted scores and the ground truth next GN. Once trained, the model can be used in real time during user

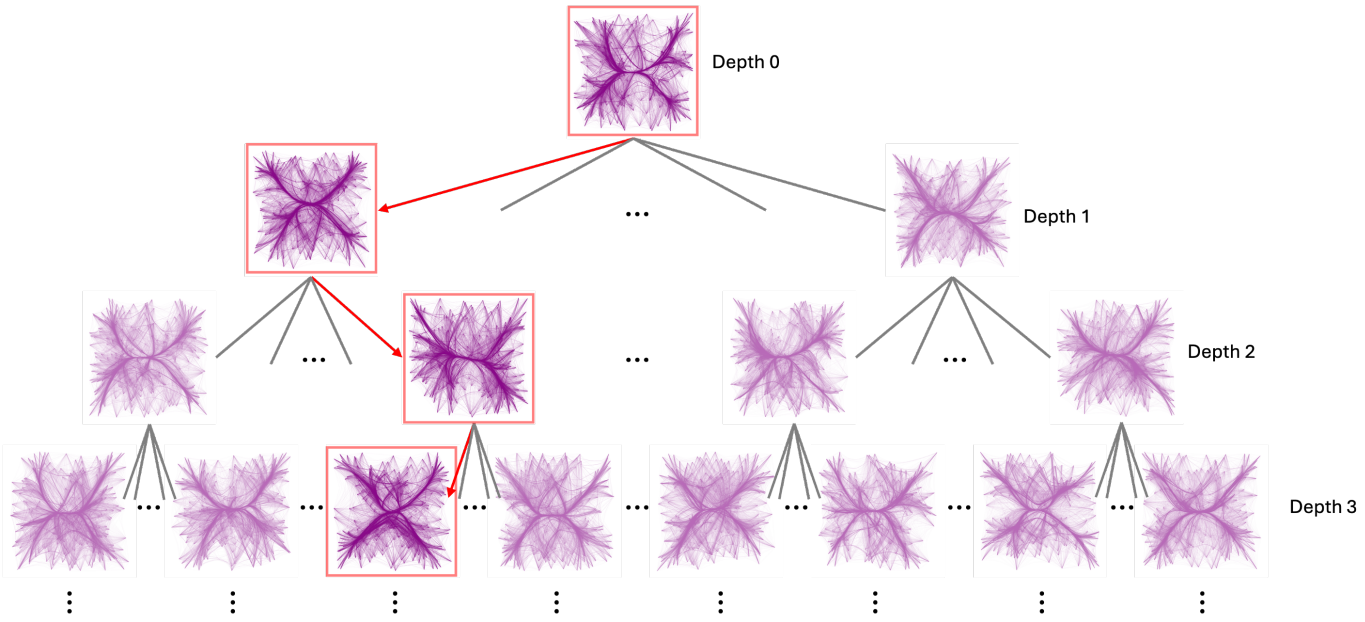


Fig. 4: Edge bundling results along a sample user trajectory in the Graph Tree (GT). Each bundled subgraph corresponds to a Graph Node (GN) visited during exploration. The red path highlights an exploratory trajectory in the Exploratory Space (ES), illustrating how bundled edges evolve as the user moves across different levels of the hierarchy. This demonstrates how our framework renders only the current GN while prefetching the next likely GN to maintain interactivity.

exploration to predict future navigation and guide prefetching. The detailed training workflow is outlined in Algorithm 1.

D. Prefetching Algorithm

In large-scale data visualization, out-of-core methods enable the processing of datasets that exceed the capacity of main memory (RAM). Instead of loading the entire dataset at once, the system streams data in chunks, keeping only the required portions in fast memory while the rest resides on slower storage such as disk, SSD, or distributed systems. For large graphs, especially when storing edge information, out-of-core processing is essential to dynamically load only the content relevant to the current visualization.

A common practice is to maintain a cache in fast memory that temporarily stores frequently accessed data blocks, exploiting spatial and temporal locality so that repeated requests can be served without disk I/O. To further reduce latency, prefetching is employed to predictively load data blocks before they are explicitly needed. In this way, required edge files are already available in RAM or GPU memory when the user interacts with the visualization, overlapping data transfer with computation. The prefetching strategy proposed in this work consists of two steps.

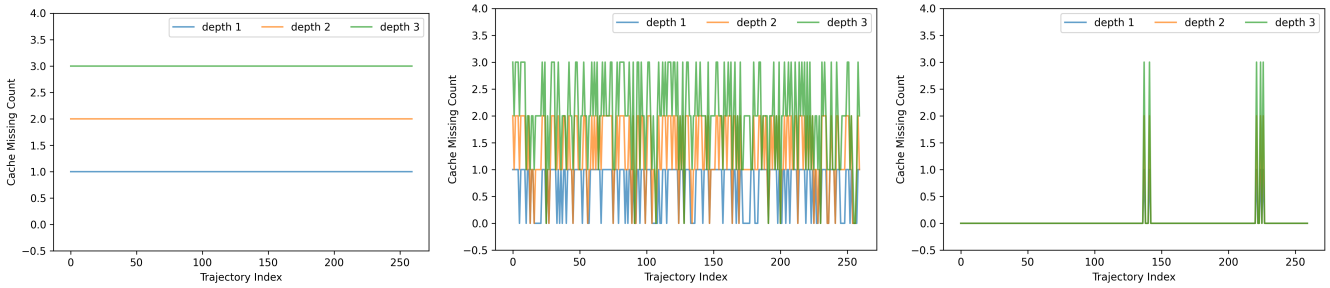
1) Prediction of the Next GN: During graph exploration within the ES, a sequence of GNs is recorded as the user navigates the GT. While the edge bundling algorithm computes the visualization for the current GN, the predictive neural network infers the most probable next GN. This inference step is executed on the GPU and overlaps entirely with edge bundling execution, since the neural network is significantly faster than the bundling process.

2) Caching of the Predicted GN: Once the predictive model outputs the next GN with the highest probability, the rendering pipeline checks whether the required data blocks are already present in the cache. If the cache hits, edge bundling proceeds immediately. If the cache misses, the missing blocks are loaded from storage into memory. Because data loading is slower than inference, situations may arise where loading is not complete when edge bundling finishes. In such cases, loading is preemptively stopped to avoid additional latency, and the process resumes during the next caching cycle. The detailed workflow of our prefetching strategy integrated with hierarchical edge bundling is outlined in Algorithm 2.

IV. Results and Evaluation

A. Datasets

Since large-scale graph datasets with the required structure are not publicly available, we generate a synthetic dataset to conduct comprehensive experiments. To construct a representative graph, we assign each node a random floating-point coordinate and explicitly create a hierarchical tree structure with detectable communities, enabling evaluation with existing community detection algorithms. Each Graph Node consists of two files: a node file and an edge file. The node file stores node IDs and their locations, while the edge file records the edge connections by listing source and target node IDs. Each GN contains 260 nodes. To evaluate scalability under extreme conditions, we construct each GN as a complete graph, where



(a) LRU (b) APPA (c) Our TP-Bundle Method

Fig. 5: Caching missing counts per depth. (a) shows the LRU-only caching. Since LRU has no predictive capability, cache misses occur consistently at each depth, resulting in frequent I/O stalls. (b) is the APPA prefetching method. APPA reduces some cache misses by prioritizing important data, but its reliance on static heuristics leads to inconsistent performance across depths. (c) shows our TP-Bundle method. By predicting continuity in user exploration, our method achieves a substantially higher cache hit rate, minimizing I/O delays and improving responsiveness compared to LRU and APPA.

every node connects to all others. While real-world graphs are typically much sparser, this design intentionally produces dense edge sets as a worst-case scenario, serving as a stress-test case that pushes the limits of computation and I/O in edge bundling under maximum load. The GT is generated with a depth of four, allowing us to test both hierarchical navigation and bundling performance under maximum load. This setup enables us to demonstrate the robustness and scalability of the proposed method in the most computationally demanding cases, while future work will extend experiments to real-world sparse networks for broader validation.

B. Experiment Setup

1) Predictive Neural Network: For the predictive neural network, the vocabulary size is set to 261, corresponding to the maximum sequence length of GNs in the testing dataset. The embedding/hidden dimension is set to 128. The multi-head self-attention module uses 8 heads, and the model stacks 3 Transformer encoder layers. A dropout rate of 0.1 is applied to mitigate overfitting. To accelerate computation, we employ a parallelized version of the force-directed edge bundling algorithm [19], [40]. The number of parallel worker threads is set to 1024.

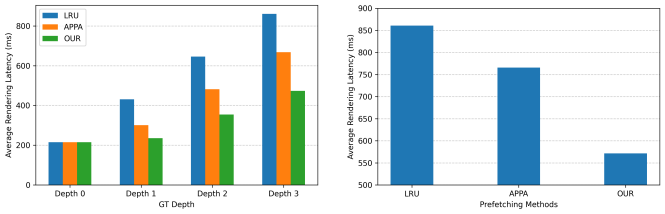
2) Training: The predictive neural network is implemented and trained using PyTorch on a single NVIDIA RTX A6000 GPU. We optimize the model using the Adam optimizer with an initial learning rate of 0.0005. Training runs for 100 epochs. The computing platform is a desktop equipped with an Intel(R) Core(TM) i7-7700K CPU (8 threads at 4.20 GHz), 32 GB DDR4 DRAM (3200 MHz), and Ubuntu 22.04.4 LTS.

C. Results

1) Rendering Quality: Figure 4 shows example rendering results for GNs at different depths. Each edge bundling image visualizes a GN, while users can select a child GN to continue exploration at deeper levels. The pipeline renders only the current GN while prefetching data for the likely next GN, thereby maintaining interactivity. The bundled edges reveal structural connection patterns within each GN and highlight the relationships between communities across levels. This hierarchical exploration enables users to progressively uncover large-scale connectivity patterns while reducing visual clutter. Importantly, the visual quality remains consistent across depths, demonstrating that the proposed pipeline effectively preserves interpretability under dense connectivity.

2) Prefetching Performance: We evaluate prefetching effectiveness by comparing our strategy against two baselines: (1) LRU-only caching, and (2) the Application-Aware Prefetching Algorithm (APPA) [49], which builds upon LRU with heuristic predictions. Cache miss counts are measured across depths 1 to 3 (depth 0 is excluded since no prior sequence exists). Results are averaged across 260 possible trajectory indices. Figures 5a–5c present cache miss counts for the three methods. LRU suffers from consistent misses at all depths, as expected. APPA reduces misses by incorporating importance-driven prediction, but its performance varies across depths. In contrast, our method achieves the lowest miss count and the highest cache hit rate, correctly predicting the majority of user navigation steps. Quantitatively, our approach reduces cache misses by up to 65% compared to LRU and 35% compared to APPA. These results confirm that the deep learning–based predictor is more effective than heuristic or purely recency-based strategies in capturing user exploration continuity.

3) Rendering Latency: We also measure rendering latency across depths for the three methods: LRU, APPA, and our TP-Bundle approach. Figure 6a compares the average latency per depth. At depth 0, all methods incur identical latency due to compulsory misses. From depths 1 to 3, latency increases for all methods due to the accumulation of edge bundling and data loading. However, our method consistently achieves the lowest latency, followed by APPA and then LRU. Figure 6b reports the overall average rendering latency aggregated across all depths. Our TP-Bundle method achieves the fastest performance, reducing latency by approximately 40% compared to LRU and 25% compared to APPA. These improvements highlight the effectiveness of combining hierarchical visualization with



(a) Across Depths (b) Overall

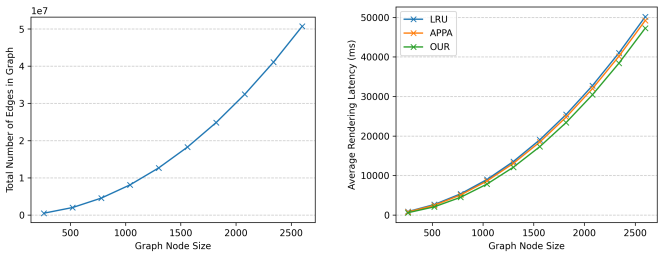
Fig. 6: Rendering latency. (a) Average rendering latency across hierarchical depths for three methods: LRU, APPA, and our TP-Bundle approach. (b) Overall average rendering latency aggregated across all hierarchical depths.

predictive prefetching: the former reduces the amount of data rendered at once, while the latter ensures that required data is already in memory when needed.

D. Scalability Test

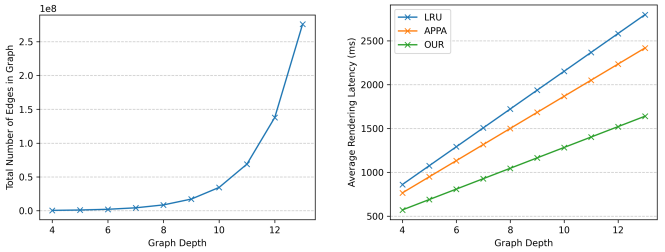
To further evaluate performance under increasing graph sizes, we conducted a scalability study by progressively enlarging the number of nodes per GN and the overall depth of the GT. For each configuration, we measured the average rendering latency across the three methods being compared. This analysis captures the combined effects of computational cost from edge bundling and I/O cost from cache misses and data transfers.

Figure 7a and Figure 8a show how the total number of edges grows with respect to the Graph Node size and Graph Tree depth. As the Graph Node grows, as shown in Figure 7b, all the prefetching methods experience exponential growth in the average rendering latency. The computational cost of edge bundling becomes the dominant factor as the Graph Node size increases. Because the proposed prefetching method achieves the lowest missing rate, it yields the lowest average rendering latency. As the Graph Tree depth grows, as shown in Figure 8b, all the prefetching methods increase linearly along with the growth of the exploratory trajectory traversing the Graph Tree. As the depth of the graph tree increases, the accuracy of the branch prediction given by the prefetching algorithm accumulates along the traversal. In this case, an accurate prefetching plays a more critical role than in the case when only the Graph Node size increases. Our TP-Bundle method can significantly reduce the average rendering latency compared to other prefetching methods because of the lowest missing ratio achieved. The baseline LRU strategy suffers from consistently high latency due to frequent cache misses. APPA provides moderate improvements but exhibits variability when user trajectories span multiple hierarchical levels. In contrast, our method maintains the lowest latency across all scales by accurately predicting exploration continuity and reducing redundant data movement. Even at extremely large graph sizes, our pipeline remains responsive, demonstrating robustness and scalability that surpass those of existing methods.



(a) Total Edges (b) Average Latency

Fig. 7: Scalability evaluation across increasing Graph Node size. (a) shows how the total number of edges in the graph grows with the graph node size. (b) shows how the average rendering latency grows with the graph node size. The proposed method achieves the lowest latency, highlighting its efficacy in handling large-scale wide tree graphs.



(a) Total Edges (b) Average Latency

Fig. 8: Scalability evaluation across increasing Graph Tree depth. (a) shows how the total number of edges in the graph grows with the graph depth. (b) shows how the average rendering latency grows with the graph depth. The proposed method achieves the lowest latency, highlighting its efficacy in handling large-scale deep tree graphs.

E. Discussion

The results demonstrate that the proposed TP-Bundle method achieves both qualitative and quantitative improvements in large-scale graph visualization. Hierarchical edge bundling enhances interpretability by reducing visual clutter, while predictive prefetching maintains responsiveness by alleviating I/O stalls. The key advantage over LRU and APPA lies in the ability of our TP-Bundle approach to capture the continuity of user exploration. APPA relies on predefined importance heuristics and static lookup tables, which cannot adapt to dynamic user behavior and often fail when users transition across multiple hierarchical levels. In contrast, our Transformer-based predictor learns sequential patterns from navigation trajectories, enabling it to anticipate common exploration behaviors. Users typically begin with a global overview, then progressively zoom into localized regions, and continue iteratively within nearby areas rather than moving arbitrarily across the graph. By aligning prefetching with this continuity, our method achieves up to 65% fewer cache misses and 40% lower rendering latency in our experimental study.

Comparison with existing accelerations. GPU- and FFT-based bundling methods primarily target computational complexity, often achieving significant speedups by parallelizing density estimation and smoothing. However, these techniques still require loading the entire dataset into GPU memory, making them susceptible to I/O bottlenecks for very large graphs. Distributed computing strategies can scale to large datasets; however, communication and synchronization overhead remain challenges when aiming for interactive responsiveness. By contrast, our approach reduces both computational and I/O costs simultaneously, achieving scalability without requiring global data residency in memory.

Generalizability. Although demonstrated on static synthetic graphs, the framework can be adapted to dynamic and multilayer networks. For example, temporal graphs could benefit from predictive prefetching by aligning cache management with expected user exploration over time slices, while multilayer or attribute-rich graphs could exploit hierarchical traversal to separate semantic dimensions. The exploratory space abstraction provides a flexible representation that could extend beyond node-link diagrams to hybrid visualization metaphors.

Limitations. While the synthetic datasets demonstrate scalability under stress-test conditions, they do not capture the sparsity or attribute complexity of many real-world networks. Future validation on domains such as social interactions, transportation systems, or biological connectomes is necessary to confirm generalizability. Furthermore, our method assumes a degree of continuity in user exploration. While this assumption holds for most exploratory workflows, abrupt or highly nonlinear navigation patterns may reduce prediction accuracy.

Overall, the evaluation indicates that integrating hierarchical organization with deep learning–based prefetching offers a promising direction for interactive graph visualization. The combination of reduced visual clutter, predictive responsiveness, and scalability under extreme edge densities demonstrates the potential of this approach to enable exploration of increasingly large and complex datasets.

V. Conclusions

In this work, we present TP-Bundle, an interactive visualization framework for exploring large-scale graphs. Our approach integrates hierarchical community detection to organize data, an exploratory space to capture user navigation, and a Transformer-based predictive prefetching algorithm to reduce I/O overhead. Comprehensive experiments demonstrate that the method consistently achieves the shortest average edge bundling latency across graph depths and trajectories, while preserving high rendering quality and supporting responsive interaction. To our knowledge, this is the first framework to combine hierarchical traversal with deep learning–based prefetching for interactive edge bundling at scale.

In the future, we plan to extend the predictive model with reinforcement learning or adaptive feedback mechanisms, which may further improve prefetch accuracy by learning from live user interactions. In addition, scaling the system across distributed architectures would allow exploration of graphs with billions of edges in real-time. Incorporating attribute-aware edge bundling and richer visual encodings could enhance analytic utility, enabling users to examine not only structural connectivity but also semantic relationships within large-scale networks. Finally, we envision applications not only in static large-scale graph visualization but also in dynamic, streaming, or multilayer networks, where predictive prefetching could fundamentally change how analysts interact with massive relational datasets.

VI. Acknowledgement

This research has been sponsored in part by the National Science Foundation grant IIS-1652846. The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] Juan V. Alemán, Santiago Pineda, et al. Stress-minimization edge bundling. In *EuroVis Short Papers*, pages 25–29, 2014.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint*, arXiv:1409.0473, 2014.

[3] Leilani Battle, Remco Chang, and Michael Stonebraker. ForeCache: A prefetching approach for interactive data visualization. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, pages 1797–1810, 2016.

[4] László A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[5] Michael Cox and David Ellsworth. Application-controlled demand paging for out-of-core visualization. In *Proceedings of IEEE Visualization (Vis)*, pages 235–244, 1997.

[6] Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong, and Xiaoming Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, 2008.

[7] Peter J. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[8] Amol Deshpande and George Karypis. Selective dissemination of information using relevance feedback. *ACM Transactions on Information Systems*, 22(1):76–112, 2004.

[9] Matthew T. Dickerson, David Eppstein, Michael T. Goodrich, and Jeremy Y. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *Journal of Graph Algorithms and Applications*, 9(1):31–52, 2005.

[10] Ozan Ersoy, Fabian Paulovich, Jo ao Poco, Luis Nonato, and Alexandru Telea. Skeleton-based edge bundling for graph visualization. *Computer Graphics Forum*, 30(3):853–862, 2011.

[11] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[12] Emden R Gansner, Yifan Hu, Stephen North, and Carlos Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *2011 IEEE Pacific Visualization Symposium*, pages 187–194. IEEE, 2011.

[13] Hanqi Guo, Ning Mao, and Xiaoru Yuan. Flow-based edge bundling for dynamic graph visualization. In *IEEE Pacific Visualization Symposium (PacificVis)*, pages 71–78, 2012.

[14] Hanqi Guo, Xiaoru Yuan, and Jian Huang. Flow-based edge bundling for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(5):786–798, 2012.

[15] Lin Guo, Wanli Zuo, Tao Peng, and Binod Kumar Adhikari. Attribute-based edge bundling for visualizing social networks. *Physica A: Statistical Mechanics and its Applications*, 438:48–55, 2015.

[16] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. In *IEEE Symposium on Information Visualization (InfoVis)*, pages 741–748, 2006.

[17] Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741–748, 2006.

[18] Danny Holten and Jarke J. van Wijk. Visual comparison of hierarchically organized data using radial edge bundles. *Computer Graphics Forum*, 27(3):495–502, 2008.

[19] Danny Holten and Jarke J. Van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[20] Christophe Hurter, Ozan Ersoy, and Alexandru Telea. Graph bundling by kernel density estimation. *Computer Graphics Forum*, 31(3):435–444, 2012.

[21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, and et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.

[22] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2018:489–504, 2018.

[23] Antoine Lambert, Romain Bourqui, and David Auber. Winding roads: Routing edges into bundles. *Computer Graphics Forum*, 29(3):853–862, 2010.

[24] Antoine Lambert, Romain Bourqui, and David Auber. Anisotropic edge bundling for continuous graphs. *Computer Graphics Forum*, 31(3):1095–1104, 2012.

[25] A. Lhuillier, C. Hurter, and A. Telea. State of the art in edge and trail bundling techniques. *Computer Graphics Forum*, 36(3):619–645, 2017.

[26] Antoine Lhuillier, Christophe Hurter, and Alexandru Telea. FFTEB: Edge bundling of huge graphs by the fast fourier transform. In *2017 IEEE Pacific visualization symposium (PacificVis)*, pages 190–199. IEEE, 2017.

[27] Jiakang Li, Songning Lai, Zhihao Shuai, Yuan Tan, Yifan Jia, Mianyang Yu, Zichen Song, Xiaokang Peng, Ziyang Xu, Yongxin Ni, Haifeng Qiu, Jiayu Yang, Yutong Liu, and Yonggang Lu. A comprehensive review of community detection in graphs. *Neurocomputing*, 600:128169, 2024.

[28] Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. In *Proceedings of IEEE Visualization (Vis)*, pages 363–370, 1996.

[29] Tallys Gustavo Martins, Nelson Lago, Higor Amario de Souza, Eduardo Felipe Zambom Santana, Alexandru Cristian Telea, and Fabio Kon. Visualizing the structure of urban mobility with bundling: A case study of the city of são paulo. *Anais*, 2020.

[30] Kresimir Matkovic, Helwig Hauser, Robert F. Tobler, and Eduard Gröller. Cache management for view-dependent volume rendering. In *Proceedings of IEEE Visualization (Vis)*, pages 345–352, 1999.

[31] Fintan McGee, Cong Guo, Alex Endert, et al. Interactive exploration of multi-layer networks with edge bundling. *Computer Graphics Forum*, 38(3):1–12, 2019.

[32] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[33] Valerio Pascucci and Kwan-Liu Ma. Visualizing very large volume datasets with the parallel particle rendering system. In *Proceedings of IEEE Visualization (Vis)*, pages 47–54, 2002.

[34] Vsevolod Peysakhovich, Christophe Hurter, and Alexandru Telea. Attribute-driven edge bundling for general graphs with applications in trail analysis. In *2015 IEEE Pacific visualization symposium (PacificVis)*, pages 39–46. IEEE, 2015.

[35] Huamin Qu, Weiwei Cui, Kui Wu, and Li Si-Ma. Mesh-based edge bundling for visualization of large graphs. *Computer Graphics Forum*, 32(3):681–690, 2013.

[36] Daniel Selassie, Jeffrey Heer, and Maneesh Agrawala. Divided edge bundling for directional network data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1099–1106, 2010.

[37] Cláudio T. Silva, Joseph S. B. Mitchell, and Peter Lindstrom. Out-of-core algorithms for scientific visualization and computer graphics. In *Proceedings of Visualization '02 Course Notes*, 2002.

[38] Jianxin Sun, David Lenz, Hongfeng Yu, and Tom Peterka. Adaptive multi-resolution encoding for interactive large-scale volume visualization through functional approximation. In *2024 IEEE 14th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 33–42, 2024.

[39] Jianxin Sun, Xinyan Xie, and Hongfeng Yu. RmdnCache: Dual-space prefetching neural network for large-scale volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 31(9):4560–4575, 2025.

[40] Yves Tuyishime, Yu Pan, and Hongfeng Yu. A distributed algorithm for force directed edge bundling. In *2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 53–54, 2020.

[41] Matthew Van Der Zwan, Valeriu Codreanu, and Alexandru Telea. CUBu: Universal real-time bundling for large graphs. *IEEE transactions on visualization and computer graphics*, 22(12):2550–2563, 2016.

[42] Ivan Viola, Armin Kanitsar, and M. Eduard Gröller. Importance-driven volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):408–418, 2006.

[43] Huamin Wu, Yingcai Wu, Fang Zhou, Wei Cui, and Huamin Qu. Multi-level density edge bundling. In *IEEE Pacific Visualization Symposium (PacificVis)*, pages 1–8, 2015.

[44] Huamin Wu, Yingcai Wu, Fang Zhou, Wei Cui, and Huamin Qu. Temporal edge bundling for dynamic graph visualization. *Computer Graphics Forum*, 35(3):1–10, 2016.

[45] Jieting Wu, Jianxin Sun, Xinyan Xie, Tian Gao, Yu Pan, and Hongfeng Yu. Accelerating web-based graph visualization with pixel-based edge bundling. In *2023 IEEE International Conference on Big Data (BigData)*, pages 6005–6014. IEEE, 2023.

[46] Jieting Wu, Lina Yu, and Hongfeng Yu. Texture-based edge bundling: A web-based approach for interactively visualizing large graphs. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2501–2508. IEEE, 2015.

[47] Jieting Wu, Jianping Zeng, Feiyu Zhu, and Hongfeng Yu. MLSEB: Edge bundling using moving least squares approximation. In *International Symposium on Graph Drawing and Network Visualization*, pages 379–393. Springer, 2017.

[48] Jieting Wu, Feiyu Zhu, Xin Liu, and Hongfeng Yu. An information-theoretic framework for evaluating edge bundling visualization. *Entropy*, 20(9):625, 2018.

[49] Lina Yu, Hongfeng Yu, Hong Jiang, and Jun Wang. An application-aware data replacement policy for interactive large-scale scientific visualization. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1216–1225, 2017.

[50] Xiaoming Yu, Pengyuan Zhou, Zhifeng Bao, and Xiaoyong Du. Application-aware prefetching for interactive data exploration. In *Proceedings of the 2018 IEEE International Conference on Data Engineering (ICDE)*, pages 1212–1223, 2018.

[51] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural Computation*, 31(7):1235–1270, 07 2019.

[52] Jianping Zeng and Hongfeng Yu. Parallel modularity-based community detection on large-scale graphs. In *2015 IEEE International Conference on Cluster Computing*, pages 1–10. IEEE, 2015.

[53] Jianping Zeng and Hongfeng Yu. A study of graph partitioning schemes for parallel graph community detection. *Parallel Computing*, 58:131–139, 2016.

[54] Jianping Zeng and Hongfeng Yu. A distributed infomap algorithm for scalable and high-quality community detection. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–11, 2018.

[55] Jianping Zeng and Hongfeng Yu. A scalable distributed louvain algorithm for large-scale graph community detection. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 268–278. IEEE, 2018.

[56] Jianping Zeng and Hongfeng Yu. Effectively unified optimization for large-scale graph community detection. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 475–482. IEEE, 2019.

[57] W. Zeng, Q. Shen, Y. Jiang, and A. Telea. Route-aware edge bundling for visualizing origin-destination trails in urban traffic. *Computer Graphics Forum*, 38(3):581–593, 2019.

[58] Wei Zhang, Kunpeng Zhang, and Philip S. Yu. Reinforcement learning based cache replacement in content delivery networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1823–1832, 2019.

[59] Hong Zhou, Weiwei Cui, Huamin Qu, Pak Chung Wong, and Xiaoming Li. Edge bundling using hierarchical control points. In *IEEE Pacific Visualization Symposium (PacificVis)*, pages 71–78, 2009.

[60] Hong Zhou, Panpan Xu, Xiaoru Yuan, and Huamin Qu. Edge bundling in information visualization. *Tsinghua Science and Technology*, 18(2):145–156, 2013.

[61] Daniel Zielasko, Benjamin Weyers, Bernd Hentschel, and Torsten W Kuhlen. Interactive 3D force-directed edge bundling. In *Computer graphics forum*, volume 35, pages 51–60. Wiley Online Library, 2016.

[62] Daniel Zielasko, Xiaoqing Zhao, Ali Can Demiralp, Torsten W Kuhlen, and Benjamin Weyers. Voxel-based edge bundling through direction-aware kernel smoothing. *Computers & Graphics*, 83:87–96, 2019.