Towards a GPU-accelerated web-based graph rendering framework for large-scale protein networks

Jiaxin Lu [1] Landon Dyken [1] Shilpika [2] Venkatram Vishwanath [2] Michael E. Papka [1,2]
Sidharth Kumar [1]

[1]University of Illinois Chicago [2]Leadership Computing Facility, Argonne National Laboratory

Introduction

- Background: Protein–protein interaction (PPI) networks are critical for understanding cellular processes. They are often large-scale and can be generated in real time from experimental or computational pipelines.
- Challenge 1: Existing frameworks (D3.js, etc) struggle with computational scalability and cannot efficiently handle real-time updates.
- Challenge 2: Current WebGPU-based systems such as GraphWaGu offer high-performance rendering but lack comprehensive APIs, biology-specific layouts (e.g., hive plots), dynamic filtering capabilities, and support for rendering directed graphs with arrowheads or other directional indicators.
- Challenge 3: Web environments lack specialized libraries for large-graph visualization, and static approaches fail to adapt to changing network density.
- Goal 1: We extend GraphWAGU to support multiple layout algorithms (hive plots, etc.), dynamic density adjustments, and a D3.js-like API for graph creation and rendering.
- Goal 2: We set new benchmarks for large-scale, real-time graph visualization in web environments, enabling biologists to interactively explore and analyze complex PPI networks through intuitive, high-performance tools.

WebGPU

- **WebGPU**: A modern web API designed to unlock modern GPU capabilities, enabling developers to perform both high-performance graphics rendering and parallel computations directly in web browsers.
- **WebGPU Computing Features**: Parallel processing capabilities, high-performance computing, cross-platform compatibility.
- **WebGPU Computing Example**: WebGPU performs matrix multiplication using parallel compute shaders, with each thread computing row-column dot products and storing results in GPU buffers, achieving CUDA-like performance.

WGSL Compute Shader Code
```
@compute @workgroup_size(16, 16)
fn main(@builtin(global_invocation_id) global_id:
vec3<u32>) {
   let row = global_id.y;
   let col = global_id.x;
   if((row < Width) && (col < Width)) {
      var Pvalue: u32 = 0;
      for(var i: u32 = 0u; i < Width; i++) {
         let m = M[row * Width + i];
         let n = N[i * Width + col];
         Pvalue = Pvalue + m * n;}
      P[row * Width + col] = Pvalue;}}
```

CUDA Code
```
__global__ void matrixMul(int *a, int *b, int *c, int N){
   int row = blockIdx.y * blockDim.y + threadIdx.y;
   int col = blockIdx.x * blockDim.x + threadIdx.x;
   int temp_sum = 0;
   if(row < N && col < N){
      for(int i = 0; i < N; i++){
         temp_sum += a[row * N + i] * b[i * N + col];}
      c[row * N + col] = temp_sum;}}
```

Figure 1. Comparison of matrix multiplication implementations: WGSL compute shader (left) and CUDA kernel (right).

- **WebGPU Rendering Features**: Modern graphics pipeline, higher frame rates (FPS), efficient resource management.
- **WebGPU in Our Work**: we use WebGPU to leverage the GPU's computational power for layout creation and rendering for directed graphs in 2 dimensions.

End-to-End Workflow of the Graph Rendering Framework

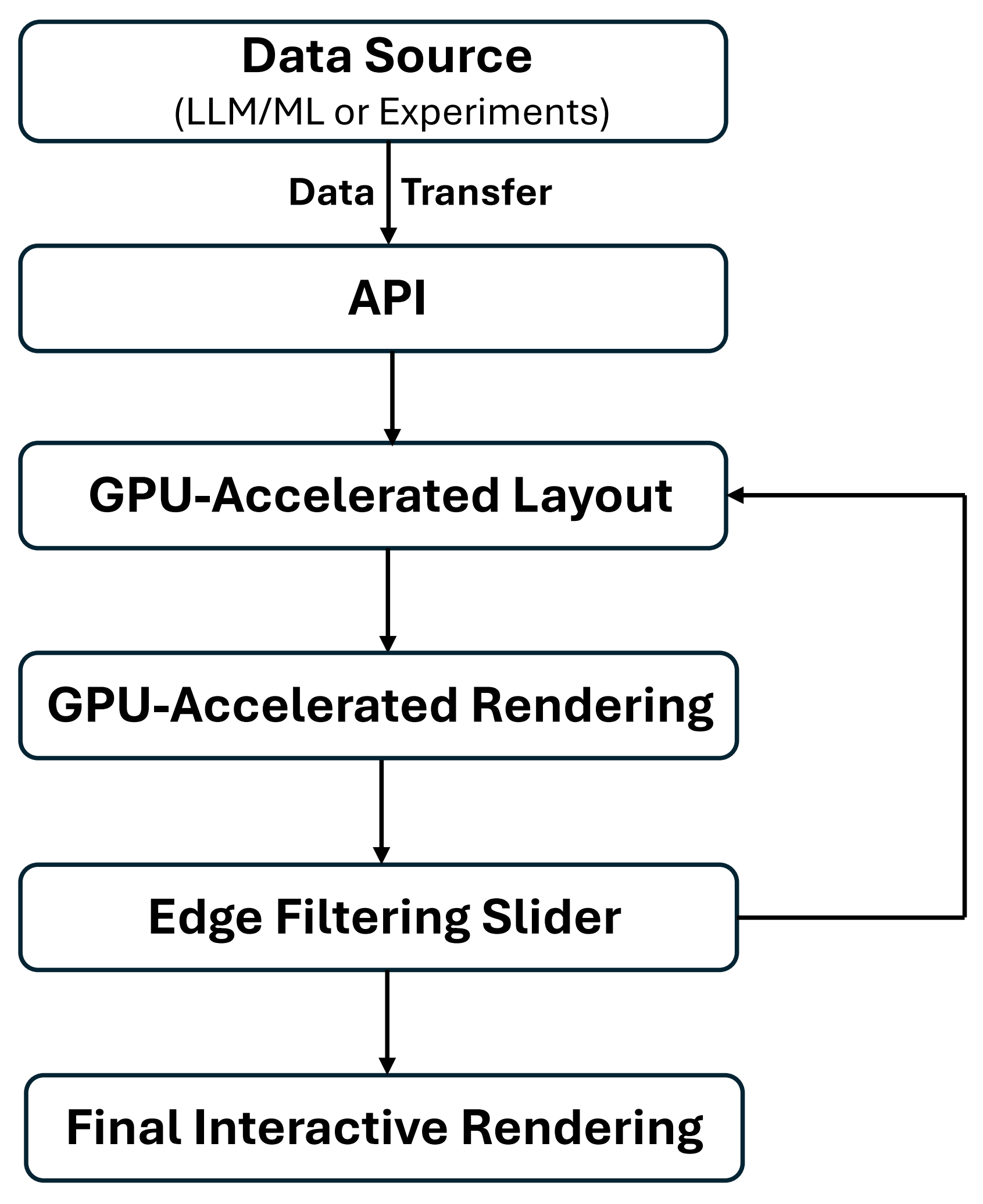

Figure 2. Workflow of our WebGPU-based Graph Rendering Framework.

- **Workflow**: Data from LLM/ML or experiments flows into the API for GPU-accelerated layout and rendering, with an interactive edge-filtering slider providing real-time, density-controlled updates.

Figure 3. Example for the graph rendering framework: raw input data (left), algorithm output (center), and detailed view of internal structure (right).

Graph Rendering API

- **Unified WebGPU API**: Provides an all-in-one interface to load, compute, and render large-scale graphs entirely in the browser, built on and extending GraphWaGu.
- **Constructor**: $new Renderer(device : GPUDevice, canvas : HTMLCanvasElement, label : HTMLLabelElement)$
- **Properties**:
  $coolingFactor$: Controls how quickly the simulation stabilizes. l: The ideal edge length.
  $theta$: The theta parameter for Barnes–Hut approximation.
  $iterationCount$: The number of iterations for the simulation. $energy$: Initial simulation energy.

Our Contributions and Extensions

Dataset and Experimental Setup:

- PPI Data Source: We utilize protein-protein interaction networks from **STRING Database (STRINGDB)**, a comprehensive repository containing functional protein associations.
- Network Characteristics: STRINGDB provides confidence-scored interactions with varying density levels, making it ideal for testing layout algorithms under different network conditions.

Challenges in Applying GraphWaGu to PPI Networks:

- **Dense connectivity**: Protein networks often contain thousands of highly interconnected nodes, creating visual clutter, and **force-directed algorithms perform poorly under extremely high edge density conditions, leading to the "hairball" effect.**
- **Our Solution**: We implement **score-based edge filtering** with confidence thresholds to **transform dense hairball networks into clear, interpretable structures.**

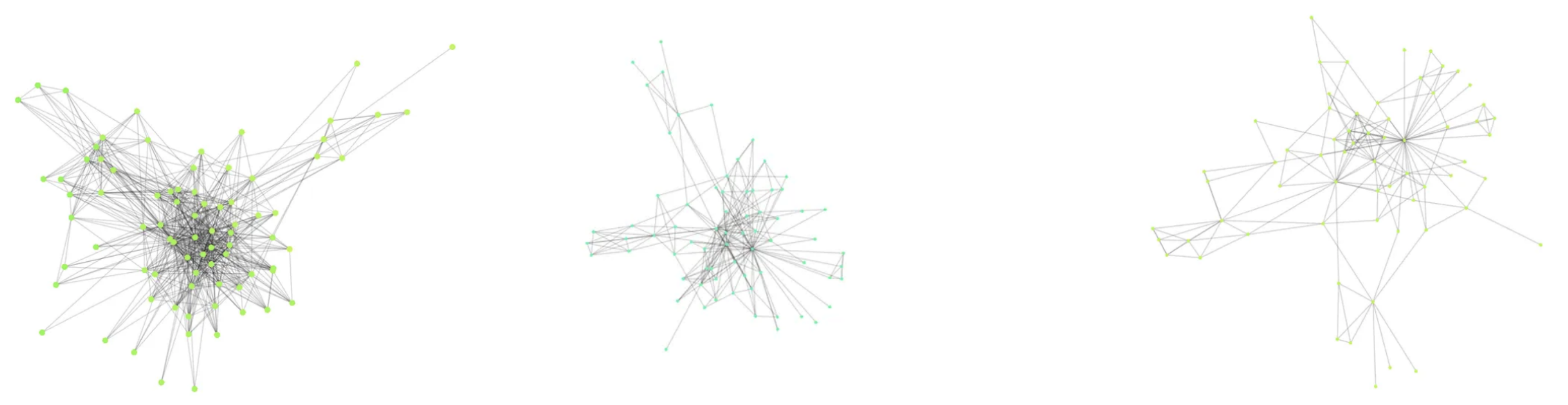

Figure 4. Progressive edge filtering with confidence thresholds: 0 (1320 edges), 0.7 (468 edges), and 0.8 (340 edges), showing transformation from hairball to interpretable network structure.

- However, this approach is static and non-interactive, lacking adaptability to different PPI networks.

- **Limited node information**: **Current GraphWaGu visualization provides insufficient protein details**, making it difficult for biologists to identify specific proteins and their functional characteristics during network exploration.
- **Our Solution**: **Degree-based visual encoding**. We render node color and size based on the degree (number of connections) of remaining nodes after filtering, enabling biologists to **immediately identify protein hubs and their relative importance** in the network.

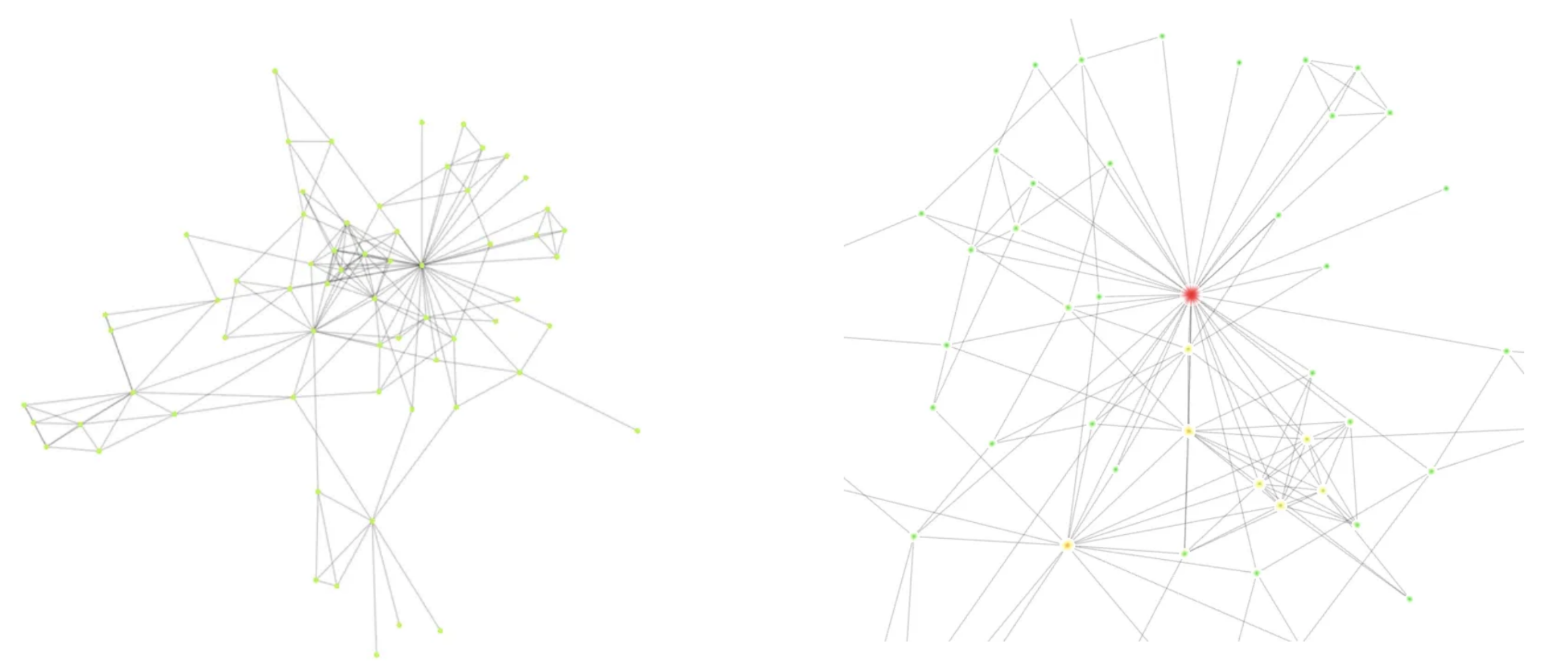

Figure 5. Visual protein importance mapping: node size and color reflect degree centrality, enabling immediate identification of high-degree protein hubs (larger, colored nodes) within the network structure.