

Evaluating LLMs on Large-Scale Graph Property Estimation via Random Walks

Sunil Kumar Maurya

University of Tokyo

Tokyo, Japan

sunil.maurya@weblab.t.u-tokyo.ac.jp

Xin Liu*

AIST

Tokyo, Japan

xin.liu@aist.go.jp

Abstract

With the rapidly improving reasoning abilities of Large Language Models (LLMs), there is also a rising demand to use them in a wide variety of domains. This brings about the need to carefully evaluate the limits of the capabilities of these models with various tests and benchmarks. Graph structures are ubiquitous in real-world data, and are often used to represent and analyze relationship patterns within data. Many benchmarks have already been proposed in the graph literature to test the reasoning ability of LLMs to follow and execute graph algorithms. However, due to the limited context length of LLMs, these benchmarks consist of very small graphs. In real-world data, the size of graphs can be significantly larger, and in many cases, not fully accessible. In this paper, we examine a class of problems that arises with very large graphs having limited accessibility. We propose a large graph benchmark dataset, EstGraph, and introduce four distinct tasks designed to estimate large graph properties. We evaluate the reasoning abilities of LLMs on these tasks using a wide variety of graph datasets. In addition, we provide task-specific prompt constructions based on random walk sampling of large graphs (up to millions of nodes) that effectively convey sufficient information to LLMs within the limits of context length. Source code and datasets are available at <https://zenodo.org/records/19632942>

1 Introduction

Large Language Models (LLMs) with reasoning abilities have opened up opportunities to utilize human-like thinking process for a wide variety of tasks and scientific domains. However, the reasoning abilities of these models are still limited. In addition, they suffer from hallucinations and can often make mistakes during multi-step operations. Therefore, researchers are making considerable efforts to benchmark the capabilities of LLMs, and

new releases of LLMs are often accompanied by evaluations on a range of benchmarks to showcase their knowledge and reasoning capabilities (Cao et al., 2025).

Similarly, there have been extensive efforts in the field of graph theory to propose new benchmarks that measure the reasoning abilities of LLMs on graph-related algorithmic tasks. Some of these benchmarks are NLGraph (Wang et al., 2023), GraphQA (Fatemi et al., 2023), GraphArena (Tang et al., 2025), GraphPattern (Dai et al., 2025), and so on. These works explore the ability of LLMs to parse graph structures described in natural language and solve graph problems. These benchmarks include questions that explore topics of local structural understanding in graphs, such as node connectivity and neighborhood identification, and global reasoning tasks, including finding shortest paths, Hamiltonian paths, or graph diameter.

Evaluations on these benchmarks show that LLMs perform well on small graphs and demonstrate graph reasoning abilities. However, as the size of the graphs grows larger, they tend to perform worse and have difficulty maintaining the context of the global structure of the graph. Figure 1 shows the performance of LLM models on a simple task of conversion from an edgelist (tuples of edges) of a graph to an adjacency list (dictionary of node as keys and list of neighbors as values). The task only requires maintaining the context of local neighborhood. However, as the size of the graphs grows, all LLM models start to miss existing edges or hallucinate new edges that do not exist in the graph. Reasoning models have fewer errors than non-reasoning models, nevertheless, the number of errors increases with the graph size. (Fatemi et al., 2023) tested various graph encoding methods for LLMs and found that even for the same graph and same graph problem, LLM performance can vary significantly based on how the graph is encoded in the input prompt (Wang et al., 2023; Finkelshtein

* Corresponding Author.

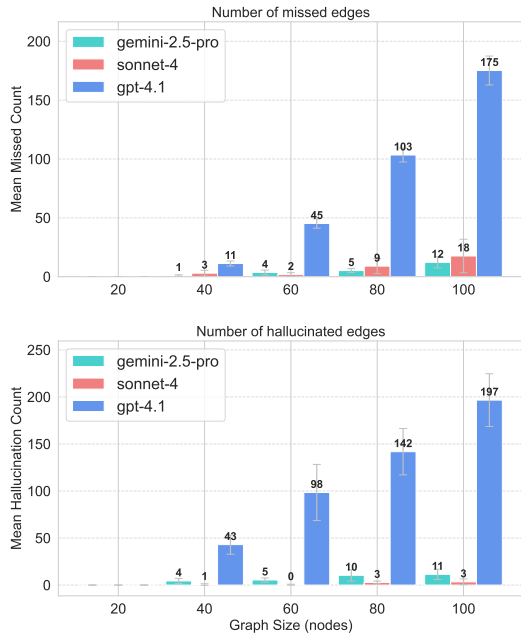


Figure 1: Plot shows number of missed edges (mean of 5 graphs for each size) and number of hallucinated edges with increase in the graph size when converting from edgelist to adjacency list for three LLM models¹.

et al., 2026). While these recent works have significantly improved our understanding of the capabilities of LLMs on graph problems, the graph sizes used in the benchmarks remain small, with only 20-50 nodes in each graph (Table 1). This limitation stems from two factors: context limits can prevent full edge information from fitting in the prompt, and even if it fits, LLMs often struggle to maintain a consistent global view of large graph structure, increasing errors.

Due to these limitations, benchmarks related to reasoning on graph structures consist of small-sized graphs, and many problem domains that require reasoning over large graphs remain underexplored. One such research area is estimation of properties of large graphs that are not fully accessible. Many real-world graphs (e.g., social, peer-to-peer, and web graphs) can have thousands to millions of nodes and are often only partially accessible via APIs. As graphs scale, attention shifts from fine-grained algorithmic details to global structure, such as community organization, connectivity patterns, and degree distributions, which are useful for analyzing information diffusion and robustness to failures or targeted attacks. Extensive research has studied estimating graph properties such as the

¹We used OpenAI’s gpt-4.1 instead of Reasoning o3 model, as it frequently refused or timed out in this experiment.

numbers of nodes/edges, cliques, and clustering coefficients from sampled subgraphs. (Gjoka et al., 2011; Katzir et al., 2011; Massoulié et al., 2006; Hardiman and Katzir, 2013).

We structure this study around the following research questions:

- How accurately can LLMs estimate global properties of large graphs using only random-walk statistics?
- How do LLMs compare with classical estimators across diverse graph structures and tasks?

Following are the contributions of our work:

- (i) We propose a large graph benchmark EstGraph and introduce four tasks to estimate large-graph properties that are relevant to real-world applications.
- (ii) We propose task-specific prompt constructions that encode the statistics about graph structure and avoid limitations related to context length. For the task of graph size estimation on real-world datasets, our approach reduces prompt length by up to 559x.
- (iii) We evaluate reasoning LLM models on graph-related estimation tasks in large graphs with up to millions of nodes, and in a real-world setting with limited access to the graphs.
- (iv) Based on our experiments, we also provide practical recommendations for large graph analysis with LLMs (Section 6 & B).

2 Related Works

With rapid improvements in LLM models, various benchmarks have been proposed to evaluate their reasoning abilities over graph-structured data. Benchmarks like GraphQA (Fatemi et al., 2023), NLGraph (Wang et al., 2023) and GraphArena (Tang et al., 2025) evaluate structure-based graph reasoning, posing questions that require inferring solutions from a graph’s topology, and share many tasks, like adjacency/neighborhood queries, connectivity or cycle detection, finding shortest paths, and so on. In contrast, GraphPattern (Dai et al., 2025) evaluates the LLMs ability to recognize primitive patterns or small-subgraph motif structures.

In these benchmarks, tasks assume a full-visibility setting, where the entire graph information is provided as a textual representation in the

Benchmark	Scale (nodes)	Access assumption	Task goal	Graph Encoding
NLGraph (Wang et al., 2023)	Max: 20	Full graph	Algorithmic execution	Edgelist text
GraphQA (Fatemi et al., 2023)	Avg: 12.3	Full graph	Algorithmic execution	Edgelist / adjacency text
GraphPattern (Dai et al., 2025)	Avg: 29.5	Full graph	Algorithmic execution	Edgelist / adjacency text
GraphArena (Tang et al., 2025)	Max: 50	Full graph	Algorithmic execution	Edgelist text
EstGraph (ours)	Max: 100k (synthetic) Max: 2.39M (real-world)	Partial graph	Estimation under partial access	Random-walk statistics

Table 1: Comparison of existing LLM-reasoning on graph benchmarks with EstGraph

prompt. An LLM is then asked to produce an exact answer to a well-defined query. This formulation is valuable for testing algorithmic reasoning on small graphs, but it becomes infeasible as the graph size grows. The sizes of graphs in these benchmarks are limited to at most 10-50 nodes (Table 1). As a consequence, existing evaluations provide limited insight into how LLMs behave when the graph is too large to be shown and only partial information can be obtained.

Random walks are used for exploring graph structure using only local neighborhood access and provide a scalable probe. They are often used as a tool for search on graphs (Bisnik and Abouzeid, 2005; Böckling et al., 2025), ranking (Brin and Page, 1998), node embedding (Perozzi et al., 2014; Grover and Leskovec, 2016), or walk-based graph representational learning (Kim et al., 2025; Ivanov and Burnaev, 2018).

Positioning of EstGraph: Existing LLM graph reasoning benchmarks primarily test exact question answering under full visibility on small graphs, whereas many practical large-graph settings require estimations from partial or limited access. Our work introduces estimation tasks over graphs with limited access and enables evaluation of LLM reasoning on graphs that are orders of magnitude larger via random walk sampling of graphs. Thus, EstGraph complements the prior graph-reasoning benchmarks and expands the evaluation to the large-scale regime.

3 Preliminaries

3.1 Definitions

Consider an undirected graph $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. The neighborhood $\Gamma(u)$ of a node u is the set of nodes v such that $u, v \in E$. The degree of a node u is

$$d(u) := |\Gamma(u)| = |\{v | \{u, v\} \in E\}|$$

and the average degree of a node u is

$$d_{avg} := \frac{1}{n} \sum_u d(u) = \frac{2m}{n}$$

The set of neighbors of node u is denoted by $N(u)$ and the degree is denoted by d_u .

Encoding the entire graph structure in a prompt is a difficult task, especially for large graphs. However, we can sample from a large graph in such a way that the resulting subgraph still provides sufficient information for the task while satisfying the context length limitations of LLMs. In the next section, we discuss the details of using random walks as an approach to sample nodes from large graphs.

3.2 Random Walks

Random walks on graphs are often used to sample nodes in graph mining applications, following connectivity patterns of the nodes. While at the current node u , the characteristics of the random walk are determined by how the next node v is chosen. For a *simple random walk*, a node v is picked uniformly at random from $\Gamma(u)$, and the walk moves to node v . The transition matrix P^{srw} associated with this random walk on G is given by

$$P^{srw}(u, v) = \begin{cases} \frac{1}{d(u)} & \text{if } \{u, v\} \in E, \\ 0 & \text{otherwise,} \end{cases}$$

The distribution of nodes and their degrees in a random walk encodes characteristics of the graph. However, random walks need to cover a sufficient portion of the graph to provide an accurate representation of the graph. Therefore, as the graph size increases, the amount of random walk data also increases linearly. As a result, directly providing random walk data in the prompt faces limitations due to the context length of LLMs. Instead, depending on the task, we can summarize the statistics of the walks that can easily scale with the graph size.

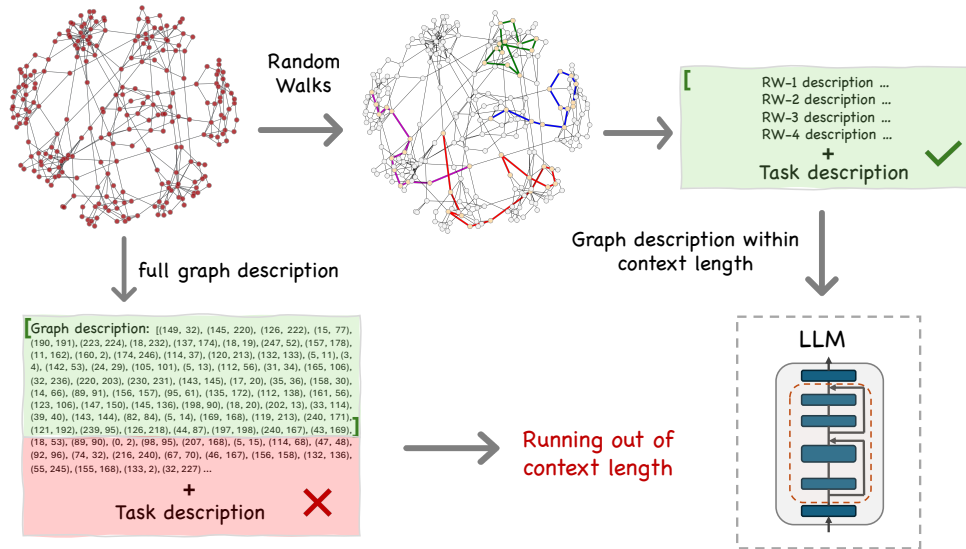


Figure 2: Figure illustrates the issue of exceeding context length as the graph size increases. Random walks on graphs provide efficient way of extracting and encoding graph-related information.

4 Estimation of Graph Properties

4.1 Estimating Number of Nodes and Edges

Estimating the size of a graph and other properties has been extensively studied in the graph literature especially for social networks like Facebook or Twitter. One common way is to use a random walk as a surrogate for uniform sampling of nodes. After sampling, we can use either *sample and collide* method which relies on the same phenomenon as the "birthday paradox" (Katzir et al., 2011; Chiericetti et al., 2016) or use the *capture-recapture* method that uses the Chapman Estimator (a corrected Lincoln-Peterson estimator) (Accettura et al., 2015). Both methods are similar, and in this work, we utilize Chapman Estimator to estimate the graph size \hat{N} ,

$$\hat{N} = \frac{(|S_1| + 1)(|S_2| + 1)}{|\mathcal{C}|} - 1 \quad (1)$$

where S_1 and S_2 are two independent sets of nodes sampled with unbiased random walks, and $|\mathcal{C}|$ is the number of common nodes in both sets.

Using the estimator, we can estimate the number of nodes by uniformly sampling two sets of nodes from the graph. To estimate the number of edges, we can find the average degree of the sampled nodes and calculate number of edges as

$\hat{M} = \frac{d_{avg} \cdot \hat{N}}{2}$. The following are some methods in the research literature that are often used to uniformly sample large graphs and estimate their properties.

Uniform Sampling from Nodelist As our first baseline, we take a simple approach of uniformly sampling nodes from the nodelist of the graph.

Metropolis-Hastings sampling This method uses a modified random walk to draw nodes from the graph such that the transition probabilities of the walk converges into a uniform distribution (Metropolis et al., 1953; Gjoka et al., 2011).

Max-degree Random Walk Sampling In this algorithm, to uniformly sample nodes via random walk, the original graph is modified into a regular graph (where all nodes have same degree) by adding self-loops to the nodes so that the degree of each node equals the maximum degree (d_{max}) of the original graph (Bar-Yossef et al., 2000; Li et al., 2015).

Return-time Based Estimation This method estimates global properties of a graph using the times of the first returns of random walks (Cooper et al., 2016).

4.2 Estimate Number of Communities

Real-world graph data often consists of clusters or communities that can arise due to social dynam-

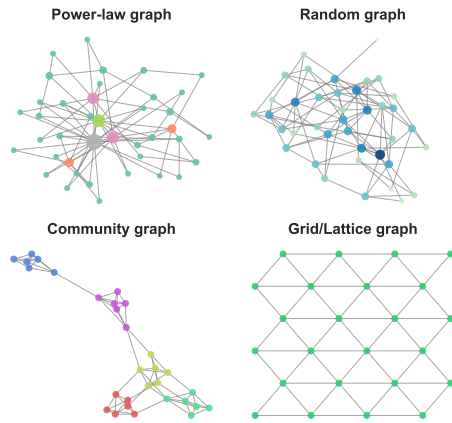


Figure 3: Graphs with varied structures

ics, geography, or some node interaction patterns depending upon the source of the data. In such graphs, a node has higher probability of edge existing between another node within same cluster than with a node from outside the cluster. This task involves detecting the number of clusters or communities in a given graph.

To benchmark community detection algorithms, we use Lancichinetti-Fortunato-Radicchi (LFR) algorithm (Lancichinetti et al., 2008) to generate synthetic graphs with a *priori* known communities and use three community detection algorithms as baselines: *Greedy Modularity* (Clauset et al., 2004), *Louvain Community Detection*, (Blondel et al., 2008) and *Label Propagation* (Cordasco and Gargano, 2010).

In our experiment to compare the performance with LLMs, instead of using the full graph, we use subgraphs induced by random walks on graphs to detect number of communities.

4.3 Identify Structure of the Graph

Real-world graphs can often exhibit a wide range of structural properties depending on the domain. These graphs are often studied to extract connectivity patterns and understand network dynamics (Yan et al., 2006; Leskovec and Horvitz, 2008). Random walks can be used to extract such information from the graphs (Rosvall and Bergstrom, 2008). In this task, we evaluate the LLMs’ capability to identify the structure of the graph by providing the degree distribution of the nodes visited in multiple random walks.

4.4 Identifying Influential Nodes in the Graph

One of the important applications of graphs in real-world data is the identification of influential nodes

in a network. These nodes often affect network dynamics by acting as critical nodes for information spread (social networks, computer networks, biological networks, etc.) or as intersections or bottlenecks (transportation networks, supply-chain networks, etc.) (Newman, 2010; Brin and Page, 1998). In this task, we test the ability of LLMs to identify node ranking measures such as *Betweenness Centrality*, *Closeness Centrality*, and *PageRank* using simple random walks. These measures are typically calculated on full graphs and are challenging to compute with partial access to the graphs.

5 Experiments

In this section, we provide details of the experimental settings with baseline graph algorithms and LLMs.

5.1 Datasets

For the task of estimating the number of nodes and edges, we generated synthetic graphs of three types: Barabasi-Albert (BA) or power-law graphs, Erdos-Renyi (ER) or random graphs, and Gaussian Random Partition (GRP) random graphs with clusters. In order to evaluate performance variations with graph scale, we divided the graphs into three categories: Small (100-1,000 nodes), Medium (1,000-10,000 nodes) and Large (10,000-100,000 nodes). In addition, we evaluated the LLMs on five real-world datasets from the SNAP graph repository, with number of nodes ranging up to millions.

For the second task of estimating the number of clusters in a graph, we used 20 LFR synthetic graphs that are often used to benchmark community detection algorithms. The number of clusters in the graphs ranges from 5 to 12.

The third task measures LLMs’ capabilities to identify the basic structure of the graph. For this task, we generated four types of synthetic graphs: BA or power-law graphs, ER graphs, LFR community graphs, and a mixture of different grid/lattice graphs. Figure 3 illustrates the structure of these graph types.

For the fourth task, we use community based 20 LFR graphs to estimate top-k ($k = 20, 50, \& 100$) high ranking nodes.

5.2 Graph Sampling

As discussed in Section 4.1, random walks are the primary way to sample from graphs. In real-world applications, simple random walks are more practical, while MH-walks (mh) or max-degree walks are

difficult to implement. Similarly, in the research literature, a *burn-in* (where a fixed number of initial nodes are dropped from the walk) is often used to lose dependence on the initial seed node; however, in practical situations, querying nodes can have an associated cost, and discarding them can be wasteful.

For our experiments, and for a fair comparison with baseline methods, we use both MH walks and srw walks with *burn-in* for generation of LLM prompts. For the task of estimating the number of clusters and graph structure, we use more practical srw walk to sample the graphs.

5.3 Prompt Generation

Previous works on benchmarking LLMs on graph-algorithmic tasks generate input prompts for LLMs by encoding the entire graph structure into natural language in the form of an edgelist or adjacency list. However, this approach fails with large graphs due to the context length limitations.

Prompting Strategy

We assume that random walk sampling in a graph provides us with the name of the node, the degree of the node, and the ability to continue the walk by accessing neighbors of the node. Based on the information derived from the walks, we create walk statistics that are relevant to the estimation task. This approach helps limit the number of tokens required in the prompt, enabling us to scale the evaluations to very large graphs.

Based on their prior knowledge about graph theory and the walk statistics in the prompt, LLMs can reason and estimate the required properties of the graphs. Figures 8-11 provide the prompt templates for all tasks. For more details on the datasets and prompt construction, please refer to Appendix E. Figure 4 shows a comparison of the number of tokens in prompts for real-world graph datasets (graph size estimation task) generated using our approach, using walk sequences in the prompt, and using edgelist to encode graphs in previous works. With our approach, the number of tokens are significantly smaller and well within LLMs' context lengths.

5.4 Results

5.4.1 Graph size estimation

Table 2 & 3 show the comparison of relative error percentages in the estimation of the number

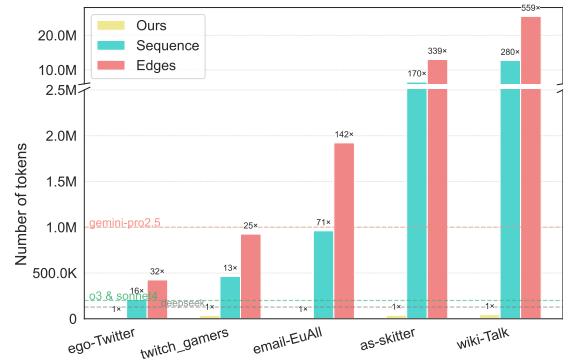


Figure 4: Figure shows comparison of number of tokens (with tiktoken) in our statistics-based prompt vs naive encoding of real-world graphs. Horizontal dashed line indicates the context length of LLM models.

of nodes. Edge estimation results are included in Appendix A.

Observations

- Conventional methods uniform sampling and mh, perform well with lower error values, while max_degree and return_walk have higher error values. However, these methods also require unbiased sampling of graphs, which is difficult in real-world applications.
- For synthetic datasets, LLM models can provide reasonable estimates of graph size with median relative error percentages below 20% in many cases. Overall, o3 performs best among all LLM models, sometimes even better than the baseline methods. For real-world much larger datasets, error percentages are higher with gemini-2.5-pro providing the best estimations, however they still outperform baseline method like mh.
- For baseline methods, mean and median error values do not differ much, implying consistent estimations over multiple graphs. However, LLM models show larger differences, suggesting higher variation in the estimated values. For a few predictions, LLMs significantly overestimate the number of nodes and edges, leading to skewed mean error values.
- Prompts for LLMs were created with two walk settings: mh and srw. srw is more practical to implement, its overall performance is lower with 78% (BA), 51% (GRP) and 9% (real-world) relative to LLMs with mh setting. Results on the estimation of edges follow similar patterns.

Graph Type	Method	Small			Medium			Large			Rank
		Median	Mean	Std	Median	Mean	Std	Median	Mean	Std	
BA	uniform [†]	13.63	13.13	6.77	4.20	5.96	4.52	0.60	0.83	0.78	1
	mh [†]	14.28	17.89	10.28	10.37	10.14	5.79	12.17	11.70	2.72	2
	max_degree [†]	25.83	27.03	9.77	27.43	27.07	3.01	27.21	27.01	0.96	8
	return_walk [†]	48.10	75.06	100.33	20.02	38.22	33.76	22.79	26.31	17.12	11
	gemini-2.5-pro (mh) [†]	20.44	32.09	30.61	40.19	52.32	46.43	20.03	43.62	50.18	9
	o3 (mh) [†]	17.60	50.31	86.82	9.63	37.64	87.75	13.08	22.17	30.76	3
	sonnet-4 (mh) [†]	23.09	35.95	43.15	12.26	18.79	18.96	30.75	32.45	22.76	5
	deepseek-v3.1 (mh) [†]	21.95	23.99	14.54	12.11	18.40	17.59	14.01	41.51	72.55	4
	gemini-2.5-pro (srw) [†]	23.32	43.65	46.38	62.02	60.05	44.42	52.56	53.74	28.78	12
	o3 (srw) [†]	16.01	42.41	62.15	30.58	37.84	30.07	25.47	24.04	9.65	6
sonnet-4 (srw) [†]	20.48	30.65	29.21	32.12	151.18	464.27	36.52	36.66	17.13	10	
deepseek-v3.1 (srw) [†]	20.23	22.09	9.97	26.87	36.13	26.30	26.97	35.38	15.55	7	
ER	uniform [†]	12.68	14.32	7.07	4.40	6.24	3.89	0.77	0.92	0.74	1
	mh [†]	13.54	15.86	10.80	6.30	6.11	4.44	2.39	2.44	1.09	4
	max_degree [†]	15.03	17.95	10.22	6.48	6.04	3.69	4.80	5.49	2.19	5
	return_walk [†]	45.86	75.96	82.72	62.29	118.98	128.25	54.87	68.69	58.08	12
	gemini-2.5-pro (mh) [†]	20.55	27.22	25.72	16.97	26.89	19.43	12.12	22.92	28.12	8
	o3 (mh) [†]	7.59	10.04	7.01	8.90	7.72	5.27	3.41	9.37	12.75	3
	sonnet-4 (mh) [†]	23.70	68.05	158.00	48.08	38.01	22.04	36.83	32.30	23.51	11
	deepseek-v3.1 (mh) [†]	9.41	16.43	16.88	4.50	13.88	21.02	4.04	10.89	14.06	2
	gemini-2.5-pro (srw) [†]	33.87	34.50	28.61	9.40	14.21	14.56	8.08	15.28	14.08	9
	o3 (srw) [†]	9.59	16.98	21.02	12.95	15.79	12.36	5.57	8.74	9.27	6
sonnet-4 (srw) [†]	13.98	21.46	17.28	25.23	29.93	24.55	19.00	25.87	20.78	10	
deepseek-v3.1 (srw) [†]	18.41	20.35	11.57	6.00	10.40	11.76	6.87	18.78	22.76	7	
GRP	uniform [†]	14.07	14.61	6.27	4.42	6.22	3.93	0.56	0.76	0.74	4
	mh [†]	7.21	10.04	8.28	4.04	6.06	5.93	2.51	2.70	1.29	1
	max_degree [†]	19.41	21.46	13.95	5.48	6.53	5.50	5.64	5.63	1.66	7
	return_walk [†]	82.10	83.87	57.52	65.99	102.77	96.60	21.41	55.97	88.00	12
	gemini-2.5-pro (mh) [†]	36.64	81.83	170.06	15.37	25.42	19.96	5.38	8.53	8.61	10
	o3 (mh) [†]	11.90	25.81	24.44	3.09	7.83	10.35	2.81	7.87	14.28	2
	sonnet-4 (mh) [†]	19.53	28.50	30.06	11.20	25.59	23.52	17.54	25.82	24.59	9
	deepseek-v3.1 (mh) [†]	15.25	19.63	14.95	3.52	9.37	16.00	3.06	10.13	15.75	5
	gemini-2.5-pro (srw) [†]	22.48	26.24	30.81	7.98	13.55	12.43	16.84	19.89	20.01	8
	o3 (srw) [†]	6.73	11.00	9.30	6.81	7.13	6.55	4.94	6.36	4.25	3
sonnet-4 (srw) [†]	23.70	26.66	20.79	12.50	21.32	20.28	44.16	40.62	22.01	11	
deepseek-v3.1 (srw) [†]	11.83	19.54	18.34	6.40	8.68	12.24	4.94	8.84	9.55	6	

Table 2: Comparison of relative error percentages for node estimations. **Bold** values show best performance among all methods and **green** values show best performance among LLM models. Ranking is based on median values. [†] methods require unbiased sampling (difficult in real-world applications); [‡] methods require srw walks.

5.4.2 Estimate Number of Communities

Figure 5 shows the mean absolute error in predicting the number of communities in the LFR graphs.

Observations

- Among baseline methods, Greedy and Louvain are able to detect communities accurately with almost no error with subgraphs created based on random walks, while Label propagation fails with significant errors.
- LLM models detected communities with error rates ranging from 1.9 to 2.6 with communities size ranging from 5 to 12 in numbers.

5.4.3 Estimate Graph Structure

Table 4 shows the accuracy of the prediction of the graph structure by LLMs.

Observations

- All LLMs were able to easily identify the grid structure due to uniform degree distribution.

- Overall, the o3 model performed best, with low accuracy only in the LFR graphs.

- For BA and LFR graphs, the o3 and gemini-2.5-pro models showed a contrast in their predictions. While o3’s predictions were biased towards BA, gemini model’s predictions were biased towards LFR. However, neither model predicted the label "ER" for any BA/LFR graphs, demonstrating their ability to distinguish between the uniform degree distribution of ER graphs and the skewed degree distribution of BA and LFR graphs (Figure 7).

5.4.4 Estimate Influential Nodes in the Graph

Table 5 shows the prediction accuracy for node ranking by LLMs.

Observations

- Betweenness and Closeness ranking measures are based on shortest-path calculations and

Method	ego-Twitter			twitch-gamers			email-EuAll			as-skitter			wiki-Talk			Rank
	Median	Mean	Std	Median	Mean	Std	Median	Mean	Std	Median	Mean	Std	Median	Mean	Std	
uniform [†]	0.83	1.06	0.81	0.52	0.69	0.53	0.68	0.80	0.66	0.17	0.21	0.16	0.10	0.18	0.14	1
mh [†]	51.02	52.04	3.84	59.62	59.77	1.10	136.20	134.75	5.87	75.21	75.40	1.27	181.04	181.16	0.41	10
gemini-2.5-pro (mh) [†]	66.04	119.25	117.18	36.64	31.29	7.52	19.06	23.03	16.44	30.01	49.82	50.01	64.37	227.64	371.45	4
o3 (mh) [†]	33.60	174.23	314.38	36.65	38.38	4.65	57.48	58.00	4.01	40.99	37.33	8.03	58.14	59.65	3.56	6
sonnet-4 (mh) [†]	33.58	37.77	9.82	40.52	42.60	18.73	62.19	61.16	7.09	49.84	44.24	23.93	64.42	61.24	8.84	8
deepseek-v3.1 (mh) [†]	33.40	36.92	7.76	36.64	36.64	0.0	57.48	52.25	9.33	42.32	52.17	13.48	64.37	64.37	0.01	7
gemini-2.5-pro (srw) [†]	44.65	205.33	320.05	52.41	67.20	37.00	29.98	70.84	65.61	32.39	55.44	59.27	27.65	98.27	111.25	2
o3 (srw) [†]	51.85	268.31	481.66	52.41	52.18	0.53	28.84	41.32	51.30	49.84	45.77	9.27	33.03	32.37	1.46	3
sonnet-4 (srw) [†]	63.10	59.66	5.73	55.39	50.48	18.45	46.63	46.63	13.34	50.43	54.17	7.56	53.95	53.37	9.86	9
deepseek-v3.1 (srw) [†]	51.83	43.50	18.61	52.41	53.60	2.66	29.99	30.71	1.52	50.21	50.21	0.00	34.38	34.38	0.00	5

Table 3: Comparison of relative error percentages for node estimations for real-world datasets. Ranking is based on median values. max_degree and return_walk are omitted due to large execution time. [†] indicate methods that are difficult to implement in real-world setting, while [†] methods can be used in real-world tasks.

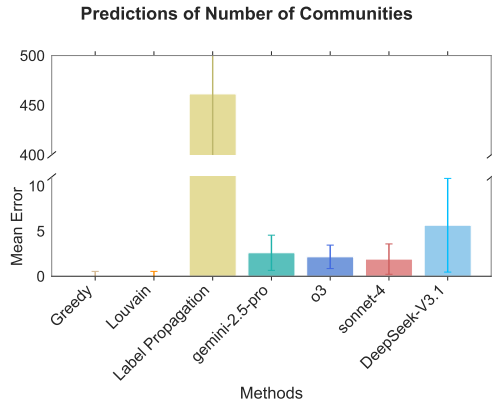


Figure 5: Figure shows the Mean error in prediction of number of communities present in the LFR graphs.

Models	BA	ER	LFR	Grid
gemini-2.5-pro	33.3%	73.3%	80.0%	100.0%
o3	93.3%	93.3%	26.7%	100.0%
sonnet-4	100.0%	13.3%	6.7%	100.0%
DeepSeek-V3.1	80.0%	66.67%	66.67%	100.0%

Table 4: Accuracy in graph structure prediction by LLMs

Model	Betweenness	Closeness	PageRank
gemini-2.5-pro	6.50 ± 7.43	9.25 ± 8.41	27.50 ± 18.41
o3	31.50 ± 14.20	35.00 ± 11.73	81.00 ± 19.85
sonnet	15.25 ± 10.06	23.75 ± 16.11	42.75 ± 28.39
DeepSeek-V3.1	23.00 ± 13.64	20.00 ± 16.43	28.50 ± 23.03

Table 5: Precision@20 (%) for node centrality values.

are therefore difficult to estimate with simple random walks. However, the o3 model can achieve a mean 30–35% precision@20, based on the degree distribution of nodes in the walks, while other models perform poorly.

- PageRank values are often correlated with how frequently a simple random walk would visit each node. Therefore, all models perform better on this ranking measure. However, the o3 model significantly outperforms other LLMs in its predictions.

6 Discussion

Large Graph Analysis with LLMs

In our experiments, we observe that LLMs can leverage random-walk statistics and the degree distribution of nodes to understand graph structure and estimate many properties of a given graph. Without any explicit instructions in the prompt, models use reasoning abilities to distinguish between different types of graphs, identify community patterns in the walks, influential nodes in the graphs, and produce graph-size estimates.

Practical recommendations for Large Graph Analysis with LLMs

The following points can be used when devising large graph analysis with LLMs:

- LLMs can provide reasonable graph size estimates from walk statistics (relative error below 20% in many cases). While unbiased sampling (mh-walk) is ideal in principle, it can be difficult to implement in practical settings, and a simple random walk can also provide good estimates for graph size.
- Using multiple runs for analysis and using median or robust aggregation of estimates helps to identify any potential outliers estimations.
- Graph size estimates generally improve as the random-walk budget increases (Figure 6).
- For community detection, longer walks can help to identify communities more easily, with repeated visits within a region providing stronger evidence of clustering. This process can also be split into multiple prompts with different seed nodes or API entry endpoints.
- LLMs use degree distributions to detect graph structure. Using multiple short-to-medium walks from diverse start nodes can help reduce local-region bias.

- (vi) Estimating PageRank-like importance is more feasible than shortest-path based ranking measures. However, model choice plays an important role with estimation accuracy.

7 Conclusion

In this work, we propose a new benchmark EstGraph and evaluate the abilities of reasoning-based LLMs on estimation tasks for large graphs that have not previously been studied using LLM reasoning. Our experiments show with appropriate prompting method, LLM models can be used to estimate properties of large graphs with reasonable accuracy.

Limitations

In our work, we have evaluated the capabilities of reasoning LLM models on the task of estimating properties of large graphs. This topic has been extensively researched with algorithmic-based estimation approaches under various settings. Although we have included a wide variety of tasks with various types of graph structures, however, our evaluation still does not cover the entire range of possible tasks.

Another potential limitation is the lack of ablation over multiple hyperparameters like sampling size, burn-in rate, random walk initializations, and so on. Due to the high cost and token consumption of reasoning LLM APIs, it was not feasible to run experiments under many different hyperparameter settings. As models continue to improve and the cost per token decreases, it may become possible to extend these experiments to cover more comprehensive settings.

Potential Risks Discussion

Our work focuses on the evaluation of LLM models to estimate graph properties from random walk statistics. The datasets used in this study are either synthetic or already publicly available. Given the current scope, our work does not pose any potential risks.

Acknowledgments

This work was partially supported by JSPS Grant-in-Aid for Scientific Research (grant number 25K03231, 23H03451). This paper is based on results obtained from the BRIDGE Program (R7-H05), implemented by the Cabinet Office, Government of Japan.

References

- Nicola Accettura, Giovanni Neglia, and Luigi Alfredo Grieco. 2015. [The Capture-Recapture approach for population estimation in computer networks](#). *Computer Networks*, 89:107–122.
- Eugenio Angriman, Alexander van der Grinten, Michael Hamann, Henning Meyerhenke, and Manuel Penschuck. 2022. [Algorithms for Large-Scale Network Analysis and the NetworKit Toolkit](#). In Hannah Bast, Claudius Korzen, Ulrich Meyer, and Manuel Penschuck, editors, *Algorithms for Big Data: DFG Priority Program 1736*, pages 3–20. Springer Nature Switzerland, Cham.
- Ziv Bar-Yossef, Alexander Berg, Steve Chien, Jittat Fakcharoenphol, and Dror Weitz. 2000. [Approximating Aggregate Queries about Web Pages via Random Walks](#). In *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB '00*, pages 535–544, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Murray A. Beauchamp. 1965. [An improved index of centrality](#). *Behavioral Science*, 10(2):161–163.
- Nabhendra Bisnik and Alhoussein Abouzeid. 2005. [Modeling and Analysis of Random Walk Search Algorithms in P2P Networks](#). In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems, HOT-P2P '05*, pages 95–103, USA. IEEE Computer Society.
- Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. [Fast unfolding of communities in large networks](#). *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008.
- Michele Borassi and Emanuele Natale. 2019. [KADABRA is an Adaptive Algorithm for Betweenness via Random Approximation](#). *J. Exp. Algorithmics*, 24(1):1.2:1–1.2:35.
- Ulrik Brandes and Christian Pich. 2007. [Centrality estimation in large networks](#). *International Journal of Bifurcation and Chaos*, 17(07):2303–2318.
- Sergey Brin and Lawrence Page. 1998. [The anatomy of a large-scale hypertextual Web search engine](#). *Computer Networks and ISDN Systems*, 30(1):107–117.
- Martin Böckling, Heiko Paulheim, and Andreea Iana. 2025. [Walk&Retrieve: Simple Yet Effective Zero-shot Retrieval-Augmented Generation via Knowledge Graph Walks](#). *arXiv preprint*. ArXiv:2505.16849 [cs].
- Yixin Cao, Shibo Hong, Xinze Li, Jiahao Ying, Yubo Ma, Haiyuan Liang, Yantao Liu, Zijun Yao, Xiaozhi Wang, Dan Huang, Wenxuan Zhang, Lifu Huang, Muhao Chen, Lei Hou, Qianru Sun, Xingjun Ma, Zuxuan Wu, Min-Yen Kan, David Lo, and 8 others. 2025. [Toward Generalizable Evaluation in the LLM Era: A Survey Beyond Benchmarks](#). *arXiv preprint*. ArXiv:2504.18838 [cs].

- Flavio Chiericetti, Anirban Dasgupta, Ravi Kumar, Silvio Lattanzi, and Tamás Sarlós. 2016. [On Sampling Nodes in a Network](#). In *Proceedings of the 25th International Conference on World Wide Web, WWW '16*, pages 471–481, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.
- Aaron Clauset, M. E. J. Newman, and Christopher Moore. 2004. [Finding community structure in very large networks](#). *Physical Review E*, 70(6):066111.
- Colin Cooper, Tomasz Radzik, and Yiannis Siantos. 2016. [Fast Low-Cost Estimation of Network Properties Using Random Walks](#). *Internet Mathematics*, 12(4):221–238. [_eprint: https://doi.org/10.1080/15427951.2016.1164100](#).
- Gennaro Cordasco and Luisa Gargano. 2010. [Community detection via semi-synchronous label propagation algorithms](#). In *2010 IEEE International Workshop on: Business Applications of Social Network Analysis (BASNA)*, pages 1–8.
- Xinnan Dai, Haohao Qu, Yifen Shen, Bohang Zhang, Qihao Wen, Wenqi Fan, Dongsheng Li, Jiliang Tang, and Caihua Shan. 2025. [How Do Large Language Models Understand Graph Patterns? A Benchmark for Graph Pattern Comprehension](#). *arXiv preprint*. ArXiv:2410.05298 [cs].
- Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. [Talk like a Graph: Encoding Graphs for Large Language Models](#). Version Number: 1.
- Ben Finkelshtein, Silviu Cucerzan, Sujay Kumar Jauhar, and Ryen W. White. 2026. [Actions Speak Louder than Prompts: A Large-Scale Study of LLMs for Graph Inference](#). In *ICLR*.
- Linton C. Freeman. 1978. [Centrality in social networks conceptual clarification](#). *Social Networks*, 1(3):215–239.
- Minas Gjoka, Maciej Kurant, Carter T. Butts, and Athina Markopoulou. 2011. [Practical Recommendations on Crawling Online Social Networks](#). *IEEE Journal on Selected Areas in Communications*, 29(9):1872–1892.
- David F. Gleich. 2015. [PageRank Beyond the Web](#). *SIAM Review*, 57(3):321–363.
- Aditya Grover and J. Leskovec. 2016. [node2vec: Scalable Feature Learning for Networks](#). *KDD*.
- Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. [Exploring network structure, dynamics, and function using NetworkX](#). In *In Proceedings of the 7th Python in Science Conference (SciPy)*, pages 11–15.
- Stephen J. Hardiman and Liran Katzir. 2013. [Estimating clustering coefficients and size of social networks via random walk](#). In *Proceedings of the 22nd international conference on World Wide Web, WWW '13*, pages 539–550, New York, NY, USA. Association for Computing Machinery.
- Sergey Ivanov and Evgeny Burnaev. 2018. [Anonymous Walk Embeddings](#). *arXiv preprint*. ArXiv:1805.11921 [cs].
- Liran Katzir, Edo Liberty, and Oren Somekh. 2011. [Estimating sizes of social networks via biased sampling](#). In *Proceedings of the 20th international conference on World wide web, WWW '11*, pages 597–606, New York, NY, USA. Association for Computing Machinery.
- Jinwoo Kim, Olga Zaghen, Ayhan Suleymanzade, Youngmin Ryou, and Seunghoon Hong. 2025. [Revisiting Random Walks for Learning on Graphs](#). *arXiv preprint*. ArXiv:2407.01214 [cs].
- Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. [Benchmark graphs for testing community detection algorithms](#). *Physical Review E*, 78(4):046110.
- Jure Leskovec and Eric Horvitz. 2008. [Planetary-scale views on a large instant-messaging network](#). In *Proceedings of the 17th international conference on World Wide Web, WWW '08*, pages 915–924, New York, NY, USA. Association for Computing Machinery.
- Jure Leskovec and Rok Sosič. 2016. [SNAP: A General-Purpose Network Analysis and Graph-Mining Library](#). *ACM Trans. Intell. Syst. Technol.*, 8(1):1:1–1:20.
- Rong-Hua Li, Jeffrey Xu Yu, Lu Qin, Rui Mao, and Tan Jin. 2015. [On random walk based graph sampling](#). In *2015 IEEE 31st International Conference on Data Engineering*, pages 927–938.
- Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. 2006. [Peer counting and sampling in overlay networks: random walk methods](#). In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing, PODC '06*, pages 123–132, New York, NY, USA. Association for Computing Machinery.
- Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. 2021. [Graph Neural Networks for Fast Node Ranking Approximation](#). *ACM Transactions on Knowledge Discovery from Data*, 15(5):78:1–78:32.
- Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. 1953. [Equation of State Calculations by Fast Computing Machines](#). *The Journal of Chemical Physics*, 21(6):1087–1092.
- Mark Newman. 2010. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA.
- Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. [DeepWalk: online learning of social representations](#).

In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '14, pages 701–710, New York, NY, USA. Association for Computing Machinery.

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

Jianheng Tang, Qifan Zhang, Yuhan Li, Nuo Chen, and Jia Li. 2025. [GraphArena: Evaluating and Exploring Large Language Models on Graph Computation](#). *arXiv preprint*. ArXiv:2407.00379 [cs].

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023. Can language models solve graph problems in natural language? In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, pages 30840–30861, Red Hook, NY, USA. Curran Associates Inc.

Gang Yan, Tao Zhou, Bo Hu, Zhong-Qian Fu, and Bing-Hong Wang. 2006. [Efficient routing on complex networks](#). *Physical Review E*, 73(4):046108.

A Supplementary Results

A.1 Graph Size Estimation

Tables 6 and 7 show the relative error percentages for edge estimations on synthetic and real-world datasets. For synthetic datasets, the performance of LLMs for ER and GRP graphs is very similar to baseline methods such as mh. For BA dataset, they show higher error values, potentially due to the skewed degree distribution of the graphs. For real-world datasets, most results follow a similar pattern as node estimation.

A.2 Estimation of Influential Nodes

Table 8 & 9 show the prediction results of node ranking for Precision@50 and Precision@100. The results are similar to Precision@20, with o3 significantly outperforming other LLM models.

B Practical Recommendations for Large Graph Analysis with LLMs

B.1 Graph Size Estimation

- Our results show that LLMs can make reasonable graph size estimates, with a median relative error below 20% on many graph datasets.
- Unbiased random walks like Metropolis-Hastings (mh-walk) random walk can lead to better performance. However, implementing

such walks is costly, especially under limited-access settings. Hence, in practical applications, simple random walks (srw) can be employed with a slight loss in estimation accuracy.

- For graph size estimation, the length of random walks should be sufficient to accumulate node collisions. In practice, we found a walk length of at least 10% of the actual graph size can provide good estimates. However, the optimal length of walk depends on the structure of the graph.
- To estimate the number of edges in a graph, LLMs first estimate the number of nodes and the average degree of the graph and calculate the number of edges based on the estimations. Therefore, edge estimations have higher errors compared to node estimations.
- Figure 6 shows that as we increase the length of random walks, node estimation errors by LLMs decrease.
- In our experiments, we found that LLMs sometimes hallucinate estimates that are orders of magnitude higher than the ground truth value. Hence, it is advisable to run the analysis a few times, and use the median or some other robust measure to detect and remove outliers.

B.2 Estimate Number of Communities

- Our experiments show that LLMs can detect the presence of multiple communities, with a mean absolute error of ≈ 2 communities (when true count is 5-12), but they are not yet reliable for precise community counts.
- For community detection, it is useful to seed random walks from diverse locations in the graph structure to increase approximate coverage. However, in real-world settings, this might be harder to achieve.
- Longer walks help to identify communities more easily, as repeated visits to the same nodes within a region provide stronger evidence of clustering. In our experiments, we relied only on the graph structure, however in real-world data, node attributes can be used to bias the walk towards nodes sharing similar attribute types.

Graph Type	Method	Median	Small Mean	Std	Medium Median	Medium Mean	Std	Large Median	Large Mean	Std	Rank
BA	uniform [†]	11.15	11.69	7.84	8.22	8.16	5.16	0.87	1.57	1.72	1
	mh [†]	15.24	23.03	25.41	12.14	14.35	8.32	13.25	12.77	2.47	2
	max_degree [†]	16.99	18.36	12.78	22.41	21.45	4.89	22.44	22.71	1.80	4
	return_walk [†]	34.14	53.57	70.90	39.04	38.63	25.64	18.73	23.90	16.94	12
	gemini-2.5-pro (mh) [†]	28.92	29.43	23.11	18.88	23.00	20.40	24.53	29.50	22.12	6
	o3 (mh) [†]	37.50	61.04	71.32	18.37	39.76	70.50	29.58	26.76	21.31	7
	sonnet-4 (mh) [†]	24.63	46.82	58.71	22.62	24.55	16.84	20.35	24.27	19.22	5
	deepseek-v3.1 (mh) [†]	35.71	38.86	21.19	23.63	29.85	25.80	26.87	39.40	34.76	8
	gemini-2.5-pro (srw) [†]	23.78	32.31	24.53	29.08	39.39	22.72	33.61	117.26	309.52	9
	o3 (srw) [†]	48.63	74.69	62.30	18.65	40.39	39.84	22.30	37.83	40.37	11
sonnet-4 (srw) [†]	8.59	49.75	86.99	19.30	223.78	765.48	30.38	42.19	35.36	3	
deepseek-v3.1 (srw) [†]	31.76	31.37	19.04	25.69	48.00	63.83	30.41	94.17	183.97	10	
ER	uniform [†]	14.20	14.65	7.06	3.94	6.08	3.93	0.70	0.89	0.78	2
	mh [†]	15.38	16.11	13.14	2.33	4.95	4.23	1.23	1.59	1.14	3
	max_degree [†]	11.40	14.10	10.48	2.29	3.04	2.38	1.50	1.84	1.22	1
	return_walk [†]	38.52	56.93	72.63	33.92	50.31	38.22	44.33	63.13	44.35	12
	gemini-2.5-pro (mh) [†]	13.26	17.08	14.10	6.94	10.16	10.35	4.07	19.42	23.64	4
	o3 (mh) [†]	11.77	13.02	7.08	7.46	8.18	5.94	5.81	10.81	12.43	5
	sonnet-4 (mh) [†]	23.21	70.59	165.41	46.35	37.50	22.32	34.30	33.10	21.22	11
	deepseek-v3.1 (mh) [†]	15.62	19.08	19.72	6.95	17.92	24.69	21.30	26.62	24.26	9
	gemini-2.5-pro (srw) [†]	11.63	19.78	17.91	13.23	16.93	14.65	2.39	4.30	5.63	6
	o3 (srw) [†]	8.67	14.71	22.24	17.23	13.97	10.25	3.71	7.42	9.48	7
sonnet-4 (srw) [†]	17.95	21.61	16.36	23.38	28.57	23.82	22.46	25.06	25.38	10	
deepseek-v3.1 (srw) [†]	23.05	27.36	21.01	10.88	14.89	14.14	5.64	25.08	28.62	8	
GRP	uniform [†]	14.28	14.07	6.61	4.53	6.05	3.84	0.53	0.80	0.83	2
	mh [†]	8.17	10.31	7.97	5.46	6.29	5.17	1.80	1.74	1.04	1
	max_degree [†]	13.38	20.93	16.61	6.47	7.03	4.80	1.55	1.58	1.27	4
	return_walk [†]	47.63	60.42	50.86	31.29	85.20	112.12	62.51	71.91	56.81	12
	gemini-2.5-pro (mh) [†]	29.72	76.25	188.73	8.21	12.89	13.31	4.17	6.89	8.34	9
	o3 (mh) [†]	22.84	22.95	22.95	6.65	11.23	9.88	4.41	10.50	14.76	7
	sonnet-4 (mh) [†]	25.04	32.57	30.10	18.53	26.29	22.24	13.91	26.59	25.41	10
	deepseek-v3.1 (mh) [†]	27.67	28.72	25.51	6.38	13.96	21.99	6.26	11.90	14.93	8
	gemini-2.5-pro (srw) [†]	12.85	16.07	14.18	12.07	14.48	16.09	2.01	6.84	9.55	5
	o3 (srw) [†]	8.19	11.06	9.22	8.47	8.04	6.42	3.38	3.54	2.36	3
sonnet-4 (srw) [†]	22.72	27.95	20.13	7.66	18.64	19.42	47.88	39.52	23.35	11	
deepseek-v3.1 (srw) [†]	14.94	21.62	16.11	6.31	16.38	21.31	6.09	16.13	23.16	6	

Table 6: Comparison of relative error percentages for edge estimations. **Bold** values show best performance among all methods and **green** values show best performance among LLM models. Ranking is based on median values. [†] indicate methods that are difficult to implement in real-world setting, while [‡] methods can be used in real-world tasks.

- While in our experiments, we used a single prompt to detect communities, for larger graphs, walk statistics from different seed nodes can be distributed across multiple prompts, and the resulting community estimates can be aggregated to produce a final prediction.

B.3 Estimate Structure of Graph

- LLMs can reliably identify grid-like graphs and distinguish between random and power-law-structured graphs.
- LLMs use degree distributions to detect graph structure. Using multiple short-to-medium walks from diverse start nodes can help reduce local-region bias.

- Estimating graph structure with LLMs can help to select or adapt a probing strategy or downstream algorithm for further graph analysis. It can also support topological characterization real-world graphs for robustness analysis and risk assessment.

B.4 Estimate Influential Nodes in the Graph

- For LLMs, identifying PageRank-like importance in graphs is easier, with the o3 model performing the best. PageRank has been used to rank web pages. However, shortest-path based node-ranking measures, such as betweenness centrality and closeness centrality are more difficult to estimate and exhibit higher errors.

Method	ego-Twitter			twitch-gamers			email-EuAll			as-skitter			wiki-Talk			Rank
	Median	Mean	Std	Median	Mean	Std	Median	Mean	Std	Median	Mean	Std	Median	Mean	Std	
uniform [†]	1.04	1.39	0.89	0.58	0.84	0.77	3.90	3.45	2.26	0.85	0.87	0.45	1.81	2.70	2.46	1
mh [†]	18.98	18.82	1.91	21.92	22.10	0.54	30.19	30.59	1.66	28.47	28.07	1.36	22.76	22.73	0.11	2
gemini-2.5-pro (mh) [†]	16.14	28.16	22.53	35.34	51.05	32.05	36.15	35.91	16.14	37.02	43.23	19.11	49.96	144.30	237.22	3
o3 (mh) [†]	55.30	128.77	190.22	25.04	34.26	26.03	43.78	42.71	11.98	35.21	35.68	25.54	35.29	31.17	18.59	4
sonnet-4 (mh) [†]	3.15	13.51	19.48	48.51	58.82	21.36	47.18	33.40	26.64	65.75	70.46	23.73	46.31	41.10	9.69	6
deepseek-v3.1 (mh) [†]	59.66	45.66	30.24	74.72	55.58	35.65	32.51	34.49	12.99	30.30	45.75	28.18	26.88	22.31	8.08	8
gemini-2.5-pro (srw) [†]	24.41	50.94	42.41	78.64	137.64	146.08	79.58	77.03	6.46	53.59	45.73	40.59	38.48	46.46	61.45	9
o3 (srw) [†]	41.55	528.84	1106.12	36.81	27.96	15.78	7.10	139.92	292.49	53.23	40.07	27.28	84.68	79.10	49.78	7
sonnet-4 (srw) [†]	25.50	30.80	33.41	47.11	52.32	43.27	26.64	33.40	28.60	53.23	56.60	31.94	44.81	37.62	24.50	5
deepseek-v3.1 (srw) [†]	41.64	51.07	38.57	38.87	43.44	32.19	130.34	88.83	58.71	53.60	47.82	31.02	27.66	61.59	49.67	10

Table 7: Comparison of relative error percentages for edge estimations for real-world datasets. **Bold** values show best performance among all methods and **green** values show best performance among LLM models. Ranking is based on median values.

Model	Betweenness	Closeness	PageRank
gemini-2.5-pro	12.50 ± 11.80	15.20 ± 8.91	27.10 ± 8.80
o3	37.40 ± 10.02	33.90 ± 9.31	88.60 ± 14.90
sonnet	23.10 ± 11.67	27.10 ± 12.34	49.90 ± 24.29
DeepSeek-V3.1	19.80 ± 14.45	25.10 ± 15.00	28.10 ± 10.27

Table 8: Precision@50 (%) for node centrality values.

Model	Betweenness	Closeness	PageRank
gemini-2.5-pro	17.20 ± 13.52	16.00 ± 4.55	28.40 ± 5.97
o3	43.25 ± 11.65	35.05 ± 9.65	85.05 ± 14.94
sonnet	26.90 ± 11.09	29.15 ± 13.68	60.70 ± 21.17
DeepSeek-V3.1	20.90 ± 14.60	21.85 ± 17.40	22.90 ± 18.80

Table 9: Precision@100 (%) for node centrality values.

- In our experiments, o3 model significantly outperformed other models, therefore, in practical applications, the choice of the LLM model can play a significant role in estimation accuracy.
- With stronger performance on PageRank-like ranking measures, LLMs are particularly well suited for social networks or web graphs.

C Supplementary Method Details

C.1 Graph Sampling with Random Walks

The following are some methods in the research literature that are often used to uniformly sample large graphs and estimate their properties. We also use these methods as baselines to evaluate the performance of LLMs.

Uniform Sampling from Nodelist As our first baseline, we take a simple approach of uniformly sampling nodes from the nodelist of the graph. By using the nodelist, we assume access to the entire graph and use this method to establish the effectiveness of the Chapman Estimator.

Metropolis-Hastings sampling This is an application of the Metropolis-Hastings algorithm for

unbiased graph sampling. It uses a modified random walk to draw nodes from the graph such that the transition probabilities of the walk converges into a uniform distribution (Metropolis et al., 1953; Gjoka et al., 2011). The transition matrix P^{MH} of the walk is defined as:

$$P^{MH}(u, v) = \begin{cases} \frac{1}{\max(d(u), d(v))} & \{u, v\} \in E, \\ 1 - \sum_{r \in \Gamma(u)} \frac{1}{\max(d(u), d(r))} & u = v, \\ 0 & \text{otherwise,} \end{cases}$$

Max-degree Random Walk Sampling In this algorithm, to uniformly sample nodes via random walk, the original graph is modified into a regular graph (where all nodes have same degree) by adding self-loops to the nodes so that the degree of each node equals the maximum degree (d_{max}) of the original graph. When the walk is at node u , the walk selects a neighboring node from $\Gamma(u)$ with a probability of $1/d_{max}$, or stays at node u with the probability of $\frac{d_{max}-d_u}{d_{max}}$. As the walk becomes equivalent to simple random walk on a regular graph, the stationary distribution of the max-degree random walk is a uniform (Bar-Yossef et al., 2000; Li et al., 2015).

Return-time Based Estimation (Cooper et al., 2016) proposed a method to estimate global properties of a graph using the times of the first returns of random walks. To estimate the number of nodes, inversely degree-biased weighted random walks are used, where the edge weight is assigned as

$$w(u, v) = \frac{1}{d(u)} + \frac{1}{d(v)}$$

Let $Z(k) = \sum_{i=1}^k Z_i$ be the time of the k th return to the vertex u ; then the number of nodes can be estimated as

$$\hat{n} = \frac{Z(k)w(u)}{2k}$$

Under this setting, walks starting from high-weight nodes can exploit structural variations in the graphs to estimate properties efficiently.

C.2 Community Detection Methods

Following community detection algorithms are used as baseline in our experiment results:

Greedy Modularity This algorithm uses the Clauset-Newman-Moore greedy modularity maximization (Clauset et al., 2004) to find the community partition with the largest modularity.

Louvain Community detection This is a heuristic method based on modularity optimization. It efficiently uncovers communities in large networks by iteratively aggregating nodes into communities (Blondel et al., 2008).

Label Propagation In this algorithm, each vertex in the network is assigned a unique label, which will be used to determine the community it belongs to. Through an iterative process, the community size is expanded as connected groups of nodes adopt the same label (Cordasco and Gargano, 2010).

C.3 Node Ranking

We use following commonly used node ranking measures:

Betweenness Centrality Betweenness centrality measures how often a node lies on the shortest paths between other pairs of nodes. A node with high betweenness can act as a bridge and has strong influence over information flow in the network (Brandes and Pich, 2007; Borassi and Natale, 2019; Maurya et al., 2021). For a given node v , it is defined as:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t , and $\sigma_{st}(v)$ is the number of those paths that pass through v .

Closeness Centrality Closeness centrality measures how near a node is to all other nodes in the graph based on shortest-path distances. Nodes with high closeness can quickly interact with or spread information to the rest of the graph or network (Beauchamp, 1965; Freeman, 1978). For a given node v in a graph, it is defined as:

$$C_C(v) = \frac{N - 1}{\sum_{u \neq v} \delta(v, u)}$$

where $\delta(v, u)$ is the shortest path distance between nodes v and u .

PageRank PageRank assigns an importance score to each node based on the principle that a

node is important if it is linked to by other important nodes. Originally designed for web page ranking (Brin and Page, 1998; Gleich, 2015), it has been widely adopted in other fields as well.

$$PR(v) = \frac{1 - \lambda}{N} + \lambda \sum_{u \in \mathcal{N}(v)} \frac{PR(u)}{|\mathcal{N}(u)|}$$

where $\lambda \in (0, 1)$ is the damping factor, $\mathcal{N}(v)$ denotes the set of neighboring nodes of v , and $|\mathcal{N}(u)|$ is the degree of node u .

D Graph Datasets for LLM Reasoning

Many graph benchmarks have been proposed to evaluate the reasoning capabilities of LLMs. Table 10 shows the statistics of the sizes of the graphs (number of nodes) present in these datasets. As each dataset has many graphs, we present the maximum or the average number of nodes as reported in the respective papers.

Table 11 shows of the statistics of the datasets in the benchmark EstGraph. While we provide a large number of graphs in the datasets for benchmarking, similar to NLGraph paper, we use a smaller subset for evaluations in our experiments due to API costs. Table 10 shows that the sizes of the graphs used in our paper are many orders of magnitude larger than those in other graph datasets. However, our paper focuses on estimation tasks that are relevant to very large graphs, where access to the whole graph is infeasible; these tasks are not applicable to very small graphs. Therefore, the range of tasks evaluated by other graph benchmarks is different from ours and are not directly comparable. Combining our approach with other graph benchmarks can provide comprehensive evaluation strategy for LLM reasoning on graph-structured data.

Dataset	Graph Size
NLGraph (Wang et al., 2023)	Maximum: 20
GraphQA (Fatemi et al., 2023)	Average: 12.3
GraphPattern (Dai et al., 2025)	Average: 29.5
GraphArena (Tang et al., 2025)	Maximum: 50
Ours (synthetic)	Maximum: 100,000
Ours (real-world)	Maximum: 2,388,953

Table 10: Size of graph datasets for LLM Reasoning. Maximum or average number of nodes in the graph dataset is reported.

Estimation Task	Graph Type	Number of Graphs
Graph Size Estimation	BA (small)	100
	BA (medium)	100
	BA (large)	100
	ER (small)	100
	ER (medium)	100
	ER (large)	100
	GRP (small)	100
	GRP (medium)	100
Number of communities	LFR	100
	ER	100
Structure Prediction	BA	100
	Grid	100
	LFR	100
Node Ranking	LFR	100

Table 11: Dataset statistics for EstGraph benchmark

E Experiment Settings

In this section, we provide the details of the experiment settings related to each task described in Section 4:

E.1 LLM Models

For our experiments, we used the following three reasoning LLM models:

- gemini-2.5-pro from Google
- sonnet-4 from Anthropic
- o3 from OpenAI (reasoning effort: high)
- DeepSeek-V3.1 (Thinking Mode)

We generated prompts for each task and used the API endpoints of the respective models. Source code and graph data used in this experiment is available at <https://zenodo.org/records/19632942>.

E.2 Estimating Number of Nodes and Edges

For this task used synthetic as well as real-world graph datasets. Details of the datasets are as follows:

E.2.1 Dataset Generation

Synthetic Graph Generation We generated three types of synthetic graphs covering a wide range of structural characteristics, using the NetworkX framework (Hagberg et al., 2008)². Table 12 provides the NetworkX functions used to generate the

²<https://networkx.org/>

random graphs. We also used NetworkKit³ (Angriman et al., 2022) for graph processing in some experiments. For each graph type and size, we generated synthetic graphs to create the dataset. The graphs are divided into three size categories: Small (100–1,000 nodes), Medium (1,000–10,000 nodes), and Large (10,000–100,000 nodes).

Real-World Datasets We use 5 real-world datasets from the SNAP graph repository (Leskovec and Sosič, 2016)⁴.

E.2.2 Prompt generation

We chose the minimum sample size to be 20% of the graph size because, for the uniform sampling baseline, a lower sample size would not yield a sufficient number of overlapping nodes for the estimation calculations. For the other methods as well, we set the sampling size to 20% of the graph size. For the random walks on the graph, we set a *burn-in* period of 10% of the graph size. A *burn-in period* is often included in graph estimation tasks during which the nodes obtained during the walk are discarded. This avoids any bias in estimations due to the choice of the initial source node. In the case of the return_walk method, we set the number of returns to 10. The prompt template for this task is shown in Figure 8.

In the prompt, we provide the following statistics of the walk:

- Length of the walk (provides the basic information of sampled nodes)
- Number of unique nodes (provides the basic information of sampled nodes)
- Number of unique edges (provides the basic information of sampled nodes)
- Time to first node collision during the walk (provides a hint of neighborhood structure or local density of the graph and how likely a walk encounters repeated nodes.)
- Number of steps when the walk returned to the initial source node (provides a hint of neighborhood structure or local density of the graph and how likely a walk encounters the source node.)
- Number of new nodes discovered for every 10% increment of the walk (provides a hint of

³<https://networkkit.github.io/>

⁴<https://snap.stanford.edu/snap/>

Graph Type	NetworkX Generating Function	Generation parameters
Barabasi-Albert Graph (BA)	barabasi_albert_graph()	$m = \text{random.randint}(3, 5)$
Erdos-Renyi Random Graph (ER)	fast_gnp_random_graph()	number of edges 5-10 times $ N $
Gaussian Random Partition Graph (GRP)	gaussian_random_partition_graph()	$s = \text{random.uniform}(0.05, 0.2) \cdot N $; $v = \frac{s}{2}$

Table 12: NetworkX functions used for generating synthetic graphs.

Dataset Name	Number of Nodes	Number of Edges	Dataset Type
as-skitter	1,694,616	11,094,209	Autonomous system graph
twitch-gamers	168,114	6,797,557	Social network
email-EuAll	224,832	340,795	E-mail network
wiki-Talk	2,388,953	4,656,682	Communication network
ego-Twitter	81,306	1,342,310	Social network

Table 13: Summary of Real-world datasets

extent of local clustering in the graph near the source node of the walk)

- Names of some of the nodes. To avoid direct leakage of node count information, we anonymized the nodes by adding a large random number to each node name. This setting mimics real-world datasets where node names are available as serial numbers.
- Nodes with the ten highest degrees in the walk (provides limited information whether high degree nodes are clustering together)
- Nodes with the ten lowest degrees in the walk (provides limited information whether low degree nodes are clustering together)
- Degree distribution of the walk (provides information on degree distribution of the walk)
- Average degree of the nodes in the walk (provides the basic information of sampled nodes)

We also provide a statistics of all walks combined at the end. Many of these statistics are used in the graph literature to study the characteristics of the graphs.

E.3 Estimating Number of Communities

E.3.1 Dataset Generation

For this task, we randomly generated LFR benchmark graphs with size ranging from 1000 to 5000 nodes. The number of communities ranged from 5 to 12. In community detection tasks, the walks originating in a community should be long enough to traverse through the community and detect repetitions of nodes in the walk. To detect all communities, we also need walks originating in every communities. Therefore, we set the maximum size of the graph dataset to 5,000 nodes.

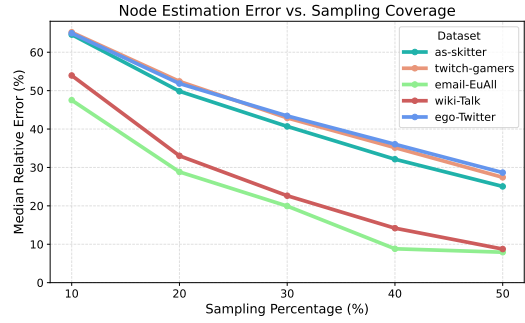


Figure 6: Graph size estimation error with sampling size (srw) for real-world datasets with o3 model.

E.3.2 Prompt Generation

To provide sufficient information to the LLMs, we select two or three initial nodes from each community based on the ground truth. The length of each simple random walk is set to 300, which can cover most community sizes. For each walk, we record the number of unique nodes, how many times each node was visited, and the degree of the nodes. We create a dictionary with nodes as keys and their number of visits and degrees as values. This process is repeated for all walks and the resulting data is added to the input prompt for the LLMs. Figure 9 shows the prompt template to create input prompts for the LLMs.

E.4 Estimate Graph Structure

E.4.1 Dataset Creation

For the task of predicting graph structures from random walks, we created synthetic datasets of four types of graphs: BA, ER, LFR, and Grid graphs. For the BA and ER graph datasets, we reused the medium-sized graphs generated in the previous task. For the LFR and Grid graphs, we generated graphs of up to 10,000 nodes. For grid graphs, we generated three types: hexagonal lattice graphs, tri-

angular lattice graphs, and hypercube graphs using NetworkX generator functions.

to those used in the previous task of estimation of graph structure.

E.4.2 Degree Distribution of Graphs

Figure 7 shows the degree distribution of graph types used in this task. The figure illustrates that the node degrees of the ER graph are spread over a wide range of values. In contrast, for BA/LFR graphs, the degree distribution is skewed, with a few nodes having very large degrees and most other nodes having small degrees. Although both LFR and BA graphs have skewed degree distributions, BA graphs (also known as power-law or scale-free graphs) exhibit significantly higher skewness than LFR graphs. When we provide degree information for various nodes in prompts to LLMs, they can distinguish between ER graphs and BA/LFR graphs.

E.4.3 Prompt Generation

To estimate the structure of a graph, random walks provide crucial information about the degree distribution. For example, in power-law graphs such as BA, the degree distribution is highly skewed, with a few nodes having very high degrees, while ER and grid graphs typically exhibit a uniform node degree distribution. For prompt construction, we perform five walks of length equal to 10% of the number of nodes in the graph. The initial seed nodes are chosen randomly. For each walk, we record the number of unique nodes, how many times each node was visited, and the degree of the nodes. We create a dictionary with nodes as keys and their number of visits and degrees as values. Figure 10 shows the template used to create input prompts for this task.

E.5 Estimating Node Ranking

E.5.1 Dataset Creation

We evaluate top- k node ranking on LFR graphs with community structure. For this task, we randomly generated LFR benchmark graphs with size ranging from 1000 to 5000 nodes. The number of communities ranges from 5 to 12. Node ranking measures like Betweenness Centrality, Closeness Centrality and PageRank were computed for all graphs and used to measure performance of LLM models.

E.5.2 Prompt Generation

For prompt construction in the node ranking estimation task, we provide graph statistics similar

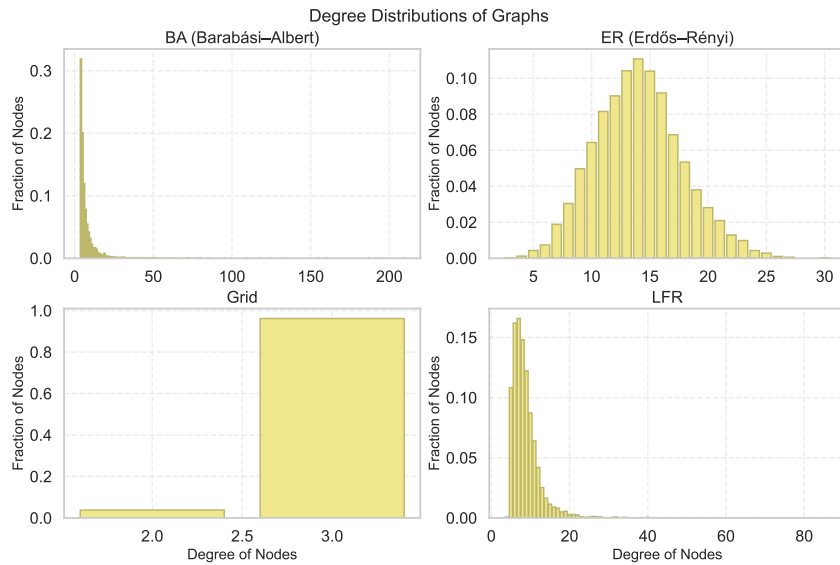


Figure 7: Figure shows the degree distribution of four types of graphs: BA, ER, Grid and LFR

Prompt template: Graph size estimation

Prompt: You are a graph theory expert. Your job is to estimate the number of nodes and edges in a large connected undirected graph based on the provided random walks. The walks are simple random walks on the graph. The random walks are used to sample nodes from the graph, and may not cover the whole graph. 2 walks are provided with associated relevant statistics. Carefully analyze the statistics of the walks. Use extensive reasoning over the provided data, and try to reduce estimation errors. Provide your final answer in the following format, Estimates:(<number of nodes>, <number of edges>)

Walk statistics are as follows:

Walk 1 statistics:

Length of walk: <>
 Number of unique nodes: <>
 Number of unique edges: <>
 First collision discovered in the walk after <> nodes.
 Walk returned to the first node after <> steps.
 Following list provides cumulative unique nodes found after roughly 10% of the walk: <>
 Following dictionary provides node occurrence frequency in the walk. Format is <number of nodes>:<number of times found in the walk. <>
 Some random nodes from the walk: <>
 Nodes with ten highest degree <>
 Nodes with ten lowest degree: <>
 Degree count of nodes in the walk: <>
 Average degree of nodes in the walk:<>

Walk 2 statistics:

...

Combined statistics of all walks:

Figure 8: Figure shows the prompt template for estimating number of nodes and edges in the graphs.

Prompt template: Community Estimation

Prompt: You are a graph theory expert. Following are statistics of simple random walks performed on a large LFR benchmark graph. The walks may not cover the whole graph. For each walk, walk statistics are provided as a dictionary: {<node name>:(<number of times node appears in the walk>, <degree of the node>),}. Use this information to predict the number of clusters or communities present in the graph. Provide your answer in the following format: Number of clusters: <number of clusters> (The number should be a single value).

Walk 1 : <dictionary of walk statistics>
Length of walk: <length of the walk>
Number of unique nodes in the walk: <unique nodes>

Walk 2 : ...
.
.
.

Figure 9: Prompt template for community estimation

Prompt template: Structure Detection

Prompt: You are a graph theory expert. Following are statistics of simple random walks performed on a large graph. The walks may not cover the whole graph. For each walk, walk statistics are provided as a dictionary: {<node name>:(<number of times node appears in the walk>, <degree of the node>), }. Use this information to predict if the graph is one of the following types: Barabasi-Albert (scale-free graph), Erdos-Renyi (random graph), LFR (Lancichinetti Fortunato Radicchi benchmark graph), or a grid graph (regular lattice). Provide your answer in the following format, Graph Type: <type of graph> (choose one from: BA, ER, LFR, Grid).

Walk 1 : <dictionary of walk statistics>
Length of walk: <length of the walk>
Number of unique nodes in the walk: <unique nodes>

Walk 2 : ...
.
.
.

Figure 10: Prompt template for the graph structure prediction

Prompt template: Top-k Nodes

Prompt: You are a graph theory expert. Following are statistics of simple random walks performed on a large graph. The walks may not cover the whole graph. For each walk, walk statistics are provided as a dictionary: {<node name>:(<number of times node appears in the walk>, <degree of the node>), }. Use this information to predict top-20 nodes that have highest Pagerank centrality values in the graph. Provide your answer in the following format, Top Nodes: <list of node names>.

Walk 1 : <dictionary of walk statistics>
Length of walk: <length of the walk>
Number of unique nodes in the walk: <unique nodes>

Walk 2 : ...
.
.
.

Figure 11: Prompt template for the top-k node prediction

You are a graph theory expert. Your job is to estimate the number of nodes and edges in a large connected undirected graph based on the provided random walks. The walks are simple random walks on the graph. The random walks are used to sample nodes from the graph, and may not cover the whole graph. 2 walks are provided with associated relevant statistics. Carefully analyze the statistics of the walks. Use extensive reasoning over the provided data, and try to reduce estimation errors. Provide your final answer in the following format, Estimates:(<number of nodes>, <number of edges>)

Walk statistics are as follows: \nWalk 1 statistics:\nLength of walk: 232
Number of unique nodes: 192
Number of unique edges: 229
First collision discovered in the walk after 20 nodes.
Walk returned to the first node after -1 steps.
Following list provides cumulative unique nodes found after roughly 10% of the walk: [22, 45, 65, 86, 103, 122, 138, 155, 174, 192].
Following dictionary provides node occurrence frequency in the walk. Format is <number of nodes>:<number of times found in the walk.> {1: 157, 2: 30, 3: 5}.
Some random nodes from the walk:
[849517,849277,849990,849577,849495,849652,849454,849245,849716,850200,849308,849647,850040,849791,849679]. The name of nodes may not start from 0, but possible to have sequentially generated.
Nodes with ten highest degree. \n Keys are degree values and list is name of the nodes. {762: 29, 662: 27, 929: 27, 656: 27, 357: 26, 63: 26, 308: 25, 326: 25, 897: 25, 957: 24}
Nodes with ten lowest degree. \n Keys are degree values and list is name of the nodes. {396: 10, 336: 10, 527: 10, 84: 10, 543: 10, 948: 9, 479: 9, 112: 9, 819: 9, 347: 9}
Degree count of nodes in the walk: {17: 27, 16: 22, 19: 18, 15: 17, 20: 15, 14: 14, 21: 12, 18: 10, 11: 9, 13: 8, 22: 7, 12: 7, 10: 6, 9: 5, 25: 3, 23: 3, 24: 3, 27: 3, 26: 2, 29: 1}
Average degree of nodes in the walk: 17.0
Walk 2 statistics:
Length of walk: 224
Number of unique nodes: 192
Number of unique edges: 222
First collision discovered in the walk after 20 nodes.
Walk returned to the first node after -1 steps.
Following list provides cumulative unique nodes found after roughly 10% of the walk: [21, 42, 60, 81, 99, 117, 137, 152, 171, 192].
Following dictionary provides node occurrence frequency in the walk. Format is <number of nodes>:<number of times found in the walk.> {1: 165, 2: 23, 3: 3, 4: 1}.
Some random nodes from the walk:
[849366,849500,849861,849499,849333,849479,849978,849447,849798,849991,849782,850027,849682,849516,849700]. The name of nodes may not start from 0, but possible to have sequentially generated.
Nodes with ten highest degree. \n Keys are degree values and list is name of the nodes. {255: 26, 259: 25, 108: 25, 892: 25, 745: 25, 89: 25, 849: 24, 9: 24, 427: 24, 379: 24}
Nodes with ten lowest degree. \n Keys are degree values and list is name of the nodes. {476: 11, 914: 11, 417: 11, 768: 10, 653: 10, 171: 10, 213: 9, 285: 9, 113: 9, 707: 6}
Degree count of nodes in the walk: {16: 23, 18: 22, 17: 22, 19: 17, 20: 16, 15: 15, 21: 14, 13: 13, 14: 11, 23: 7, 12: 6, 25: 5, 22: 5, 11: 4, 24: 4, 9: 3, 10: 3, 26: 1, 6: 1}
Average degree of nodes in the walk: 17.3
Combined statistics of all walks:
Length of walk: 456\nNumber of unique nodes: 348
Some random nodes from the walk:
[849479,850093,849454,849798,849960,849717,849833,850170,850055,849623]. The name of nodes may not start from 0, but possible to have sequentially generated.
Nodes with ten highest degree. \n Keys are degree values and list is name of the nodes. {603: 29, 950: 27, 425: 27, 502: 27, 568: 26, 310: 26, 8: 25}
Nodes with ten lowest degree. \n Keys are degree values and list is name of the nodes. {25: 9, 199: 9, 632: 9, 128: 9, 251: 9, 625: 9, 190: 9, 238: 6}
Degree count of nodes in the walk: {17: 53, 16: 52, 19: 49, 20: 40, 18: 40, 15: 34, 21: 32, 14: 29, 13: 24, 11: 16, 22: 14, 23: 13, 12: 13, 25: 10, 10: 10, 9: 9, 24: 8, 27: 3, 26: 3, 29: 2, 6: 2}
Average degree of nodes in the walk: 17.3\nNumber of shared nodes between the two walks: 36\n

Figure 12: Prompt example for graph size estimation

You are a graph theory expert. Following are statistics of simple random walks performed on a large graph. The walks may not cover the whole graph.\nFor each walk, walk statistics are provided as a dictionary: {<node name>:(<number of times node appears in the walk>, <degree of the node>), }.\nUse this information to predict if the graph is one of the following types: Barabasi-Albert (scale-free graph), Erdos-Renyi (random graph), LFR (Lancichinetti\u2013Fortunato\u2013Radicchi benchmark graph), or a grid graph (regular lattice).\nProvide your answer in the following format, Graph Type: <type of graph> (choose one from: BA, ER, LFR, Grid).

Walk 1: {5:(2,137), 7:(2,122), 12:(1,88), 10:(3,75), 19:(1,60), 8:(1,58), 41:(1,57), 25:(1,56), 43:(2,56), 33:(1,50), 13:(1,48), 26:(1,47), 54:(2,44), 16:(1,43), 34:(5,42), 3:(1,42), 14:(1,40), 28:(1,35), 23:(1,34), 39:(1,34), 4:(2,33), 47:(1,33), 31:(1,31), 51:(1,31), 62:(1,28), 58:(2,28), 24:(3,26), 20:(2,25), 65:(2,24), 230:(1,23), 30:(1,21), 44:(1,19), 38:(1,19), 110:(1,19), 281:(2,19), 123:(1,18), 83:(1,16), 140:(1,16), 214:(1,16), 153:(1,16), 86:(1,16), 283:(1,15), 94:(1,15), 219:(1,15), 90:(1,14), 302:(1,14), 231:(1,14), 472:(1,13), 128:(1,13), 176:(3,12), 368:(1,12), 139:(1,11), 254:(1,11), 377:(2,11), 194:(1,11), 295:(1,10), 389:(1,10), 413:(1,10), 70:(1,10), 516:(1,10), 606:(2,10), 312:(1,10), 744:(1,10), 87:(1,9), 480:(1,9), 63:(1,9), 421:(1,9), 462:(1,9), 367:(1,9), 980:(1,8), 616:(1,8), 284:(1,8), 292:(1,8), 947:(1,7), 681:(1,7), 709:(1,7), 1006:(1,7), 244:(1,7), 1368:(1,7), 529:(1,7), 534:(1,7), 496:(1,7), 304:(1,7), 508:(1,6), 592:(1,6), 599:(1,6), 105:(2,6), 406:(1,6), 507:(1,6), 323:(1,6), 1242:(1,6), 833:(1,6), 314:(1,6), 443:(1,5), 436:(1,5), 1452:(1,5), 1035:(2,5), 935:(1,5), 1347:(1,5), 321:(1,5), 677:(1,5), 605:(1,5), 1058:(1,5), 1084:(1,5), 641:(1,5), 1076:(2,5), 1182:(2,5), 766:(1,4), 862:(1,4), 1507:(1,4), 1494:(3,4), 1481:(1,4), 1161:(1,4), 906:(1,4), 1190:(1,4), 767:(1,4), 1348:(1,4), 1044:(1,4), 836:(1,4), 1201:(1,4), 1250:(1,4), 236:(1,4), 1337:(1,4), 1268:(1,4), 541:(1,4), 1299:(1,4), 787:(1,4)}\nLength of walk: 154. Number of unique nodes in the walk: 127\nWalk 2: {5:(1,137), 0:(3,127), 19:(2,60), 8:(1,58), 41:(1,57), 43:(3,56), 18:(2,56), 13:(1,48), 16:(1,43), 34:(2,42), 14:(1,40), 28:(2,35), 39:(2,34), 4:(1,33), 45:(2,31), 111:(3,31), 42:(1,30), 22:(1,27), 20:(1,25), 60:(4,23), 46:(2,23), 118:(1,22), 306:(2,20), 169:(1,20), 15:(1,20), 61:(1,19), 183:(1,18), 35:(2,18), 220:(1,16), 153:(1,16), 76:(1,15), 283:(2,15), 231:(3,14), 335:(1,14), 147:(1,14), 130:(1,14), 90:(1,14), 181:(1,13), 286:(1,13), 218:(1,12), 210:(1,12), 311:(1,12), 435:(1,12), 368:(1,12), 167:(1,11), 98:(1,11), 373:(1,11), 185:(1,11), 72:(1,11), 383:(1,10), 197:(1,10), 333:(2,10), 619:(2,9), 1037:(1,9), 348:(1,8), 494:(1,8), 292:(1,8), 170:(1,8), 980:(1,8), 134:(1,8), 769:(1,8), 564:(1,7), 549:(2,7), 1072:(1,7), 1158:(1,7), 487:(1,7), 407:(1,7), 250:(1,7), 941:(1,7), 224:(1,7), 233:(1,7), 238:(1,7), 465:(1,7), 310:(1,7), 513:(1,6), 794:(2,6), 997:(1,6), 643:(1,6), 658:(1,6), 1271:(1,6), 738:(1,6), 937:(3,6), 943:(1,5), 1067:(1,5), 1036:(1,5), 566:(1,5), 863:(1,5), 341:(1,5), 327:(1,5), 835:(1,5), 482:(1,5), 1321:(1,5), 1307:(1,5), 1003:(1,5), 628:(1,5), 707:(1,5), 440:(1,5), 763:(1,5), 1275:(1,5), 1262:(1,5), 1195:(1,5), 1418:(1,5), 1301:(1,5), 1521:(2,4), 554:(1,4), 1155:(1,4), 1498:(1,4), 961:(1,4), 1429:(1,4), 901:(1,4), 381:(1,4), 1397:(1,4), 1387:(1,4), 851:(1,4), 1351:(1,4), 1342:(1,4), 1292:(1,4), 1289:(1,4), 1265:(1,4), 1256:(1,4), 1250:(1,4), 735:(1,4), 1520:(1,4), 1226:(1,4), 1156:(1,4), 734:(1,4)}\nLength of walk: 154. Number of unique nodes in the walk: 126\nWalk 3: {5:(1,137), 0:(2,127), 9:(1,76), 10:(1,75), 43:(1,56), 13:(1,48), 26:(2,47), 23:(1,34), 47:(1,33), 29:(1,32), 31:(1,31), 51:(2,31), 58:(1,28), 62:(1,28), 22:(2,27), 24:(1,26), 66:(2,25), 85:(2,23), 107:(1,23), 91:(1,23), 46:(2,23), 97:(1,20), 169:(1,20), 38:(1,19), 135:(2,19), 123:(1,18), 129:(1,18), 180:(2,17), 198:(1,17), 562:(1,16), 452:(1,16), 231:(1,14), 319:(1,13), 551:(2,13), 472:(2,13), 115:(1,13), 349:(1,13), 368:(6,12), 176:(2,12), 775:(2,12), 150:(1,12), 559:(4,12), 594:(1,11), 410:(5,11), 569:(1,11), 374:(1,11), 173:(1,10), 232:(3,10), 261:(1,10), 294:(2,10), 336:(1,10), 371:(1,9), 345:(1,9), 672:(1,9), 87:(1,9), 63:(1,9), 761:(1,9), 171:(1,8), 1028:(1,8), 392:(1,8), 800:(1,7), 223:(1,7), 556:(1,7), 534:(2,7), 477:(1,7), 142:(1,7), 971:(2,6), 937:(1,6), 422:(1,6), 1018:(1,6), 657:(3,6), 662:(1,6), 785:(1,6), 783:(1,6), 1274:(1,6), 756:(1,6), 187:(1,6), 1293:(1,5), 752:(1,5), 1338:(1,5), 450:(1,5), 423:(1,5), 1076:(2,5), 933:(1,5), 359:(1,5), 669:(1,5), 1154:(1,5), 874:(1,5), 1090:(1,5), 1414:(1,5), 628:(2,5), 985:(1,4), 1332:(1,4), 1496:(1,4), 1336:(1,4), 1014:(1,4), 983:(1,4), 994:(1,4), 1033:(2,4), 1514:(1,4), 1393:(1,4), 1331:(1,4), 812:(2,4), 1494:(3,4), 981:(1,4), 1484:(1,4), 806:(1,4), 1308:(1,4), 1299:(2,4), 861:(1,4), 1408:(1,4), 674:(2,4), 1423:(1,4), 1166:(1,4)}\nLength of walk: 154. Number of unique nodes in the walk: 114\nWalk 4: {0:(1,127), 9:(1,76), 19:(3,60), 8:(1,58), 41:(1,57), 25:(2,56), 13:(1,48), 26:(1,47), 54:(1,44), 28:(2,35), 39:(2,34), 23:(2,34), 4:(1,33), 29:(1,32), 111:(3,31), 31:(1,31), 22:(1,27), 113:(2,26), 64:(1,26), 66:(3,25), 91:(1,23), 118:(1,22), 178:(1,21), 99:(2,20), 97:(1,20), 306:(1,20), 35:(1,18), 80:(1,17), 75:(1,17), 220:(1,16), 96:(1,16), 83:(1,16), 154:(1,15), 332:(1,14), 147:(1,14), 551:(1,13), 239:(1,13),

Prompt example for graph structure prediction (continued)

```

210:(2,12), 266:(1,12), 334:(1,11), 120:(1,11), 515:(1,11), 757:(1,11), 374:(1,11), 543:(1,11),
408:(2,11), 336:(1,10), 383:(1,10), 246:(1,10), 489:(1,10), 462:(1,9), 391:(1,9), 421:(1,9),
161:(1,9), 418:(1,9), 355:(1,9), 1028:(1,8), 624:(1,8), 510:(1,8), 822:(1,8), 457:(1,7),
407:(1,7), 571:(1,7), 361:(5,7), 244:(1,7), 837:(1,7), 310:(1,7), 534:(1,7), 148:(1,7),
575:(1,6), 950:(3,6), 591:(1,6), 599:(2,6), 928:(1,6), 422:(1,6), 1016:(1,6), 996:(1,6),
507:(1,6), 991:(1,6), 776:(1,6), 988:(1,6), 1186:(2,5), 604:(1,5), 916:(1,5), 1091:(1,5),
952:(1,5), 752:(1,5), 1154:(1,5), 1258:(1,5), 443:(1,5), 829:(1,5), 1182:(1,5), 618:(1,5),
308:(1,5), 1139:(2,5), 814:(1,5), 1471:(1,4), 1033:(2,4), 1034:(2,4), 1508:(1,4), 964:(1,4),
1477:(1,4), 1461:(1,4), 1446:(1,4), 985:(1,4), 1354:(1,4), 1439:(1,4), 1435:(1,4), 753:(1,4),
1227:(2,4), 1218:(1,4), 1213:(1,4), 692:(1,4), 1187:(2,4), 1297:(1,4), 1312:(1,4), 1169:(1,4),
1327:(1,4), 1331:(1,4), 1126:(2,4), 1337:(1,4), 1119:(1,4), 1415:(1,4), 912:(1,4), 1069:(1,4),
1223:(1,4)}\nLength of walk: 154. Number of unique nodes in the walk: 126\nWalk 5: {5:(1,137),
0:(1,127), 7:(4,122), 12:(2,88), 10:(1,75), 8:(1,58), 41:(1,57), 18:(2,56), 33:(1,50), 26:(2,47),
54:(1,44), 16:(2,43), 34:(1,42), 14:(2,40), 6:(1,38), 23:(1,34), 51:(1,31), 32:(2,28), 22:(1,27),
48:(1,26), 64:(1,26), 57:(1,25), 78:(1,25), 65:(2,24), 60:(2,23), 46:(1,23), 107:(2,23),
93:(3,22), 30:(1,21), 178:(2,21), 99:(1,20), 15:(3,20), 155:(1,19), 71:(1,17), 96:(1,16),
140:(1,16), 206:(1,15), 219:(1,15), 130:(1,14), 49:(1,14), 205:(2,14), 239:(1,13), 119:(1,13),
181:(1,13), 375:(1,12), 150:(1,12), 227:(1,11), 141:(1,11), 98:(1,11), 212:(1,11), 331:(2,10),
261:(2,10), 459:(1,10), 603:(1,9), 712:(1,9), 619:(1,9), 392:(2,8), 616:(1,8), 444:(2,8),
320:(1,8), 567:(2,8), 171:(1,8), 292:(1,8), 830:(1,7), 397:(2,7), 412:(1,7), 535:(1,7),
728:(1,7), 570:(1,7), 223:(1,7), 329:(1,7), 865:(1,6), 592:(1,6), 1406:(2,6), 1048:(2,6),
620:(2,6), 649:(2,6), 808:(1,6), 587:(1,5), 970:(1,5), 1418:(1,5), 1284:(1,5), 904:(1,5),
1301:(1,5), 916:(1,5), 580:(1,5), 1306:(1,5), 468:(1,5), 621:(1,5), 491:(1,5), 423:(1,5),
1363:(1,5), 1454:(1,5), 1123:(1,5), 829:(1,5), 637:(1,5), 554:(1,4), 1523:(1,4), 1079:(1,4),
1116:(1,4), 1498:(1,4), 964:(1,4), 1457:(1,4), 1451:(1,4), 1442:(1,4), 1433:(2,4), 909:(1,4),
1417:(1,4), 892:(1,4), 871:(1,4), 1362:(1,4), 828:(1,4), 778:(1,4), 753:(1,4), 237:(1,4),
734:(2,4), 1216:(1,4), 1213:(1,4), 1201:(1,4), 1199:(1,4), 1518:(1,4), 1167:(1,4), 1129:(1,4),
1190:(1,4)}\nLength of walk: 154. Number of unique nodes in the walk: 124\n"

```

Figure 13: Prompt example for graph structure prediction

You are a graph theory expert. Following are statistics of simple random walks performed on a large LFR benchmark graph. The walks may not cover the whole graph. For each walk, walk statistics are provided as a dictionary: {<node name>:(<number of times node appears in the walk>, <degree of the node>), }. Use this information to predict the number of clusters or communities present in the graph. Provide your answer in the following format: Number of clusters: <number of clusters> (The number should be a single value).

Walk 1: {597:(8,42), 443:(5,17), 65:(2,15), 83:(3,14), 77:(1,14), 893:(2,14), 92:(2,13), 581:(5,13), 574:(3,13), 54:(4,13), 552:(4,13), 526:(1,13), 60:(2,12), 166:(4,12), 530:(4,12), 517:(5,12), 201:(2,12), 140:(2,11), 134:(2,11), 638:(1,11), 390:(3,11), 622:(1,11), 171:(4,11), 178:(3,11), 80:(4,11), 572:(1,11), 927:(2,11), 607:(3,10), 99:(5,10), 614:(1,10), 113:(4,10), 861:(7,10), 121:(3,10), 583:(5,10), 640:(2,10), 66:(3,10), 61:(1,10), 559:(3,10), 4:(1,10), 0:(4,10), 421:(2,10), 435:(5,10), 107:(2,9), 615:(1,9), 169:(1,9), 592:(4,9), 683:(3,9), 415:(3,9), 999:(10,9), 109:(2,9), 191:(5,9), 76:(3,9), 403:(2,9), 898:(2,9), 355:(1,9), 626:(1,9), 206:(1,8), 16:(1,8), 97:(1,8), 90:(4,8), 600:(3,8), 395:(3,8), 926:(2,8), 871:(5,8), 200:(1,8), 114:(1,8), 47:(3,8), 966:(1,8), 777:(2,8), 56:(1,8), 115:(1,8), 655:(2,8), 372:(2,7), 407:(1,7), 598:(1,7), 941:(1,7), 961:(1,7), 569:(2,7), 976:(2,7), 528:(3,7), 933:(2,7), 392:(4,7), 370:(1,7), 889:(1,7), 874:(1,7), 885:(1,7), 285:(5,7), 251:(2,7), 233:(2,7), 714:(4,7), 376:(2,7), 194:(3,7), 186:(1,7), 707:(1,7), 656:(2,7), 665:(2,7), 379:(4,6), 994:(3,6), 476:(2,6), 978:(1,6), 365:(4,6), 616:(3,6), 969:(1,6), 925:(4,6), 449:(2,6), 892:(1,6), 940:(3,6), 680:(1,6), 173:(1,6), 591:(1,6), 904:(2,6), 434:(3,5), 870:(2,5), 661:(3,5), 603:(1,5), 899:(1,5), 458:(1,5), 199:(1,5), 897:(1,5), 400:(2,5), 382:(3,5), 965:(1,5), 493:(1,5)}\nLength of walk: 301. Number of unique nodes in the walk: 123

Walk 2: {844:(1,26), 590:(4,26), 518:(3,25), 45:(5,22), 853:(1,19), 468:(1,17), 501:(5,16), 785:(3,16), 185:(1,16), 126:(2,15), 629:(2,14), 580:(2,14), 35:(3,14), 485:(1,14), 469:(5,13), 404:(2,13), 595:(1,13), 745:(1,13), 257:(1,13), 94:(1,13), 884:(2,13), 602:(1,12), 89:(2,12), 177:(1,12), 105:(1,12), 652:(1,12), 623:(3,12), 125:(2,12), 513:(3,12), 631:(3,12), 18:(3,12), 448:(1,11), 411:(3,11), 527:(1,11), 718:(1,11), 830:(1,11), 98:(2,11), 699:(3,10), 566:(1,10), 28:(1,10), 799:(2,10), 523:(2,10), 709:(1,10), 42:(1,10), 546:(1,10), 515:(1,10), 144:(3,10), 691:(1,9), 11:(1,9), 74:(1,9), 637:(1,9), 594:(1,9), 648:(2,9), 516:(1,9), 464:(1,9), 983:(4,9), 716:(2,9), 657:(2,9), 715:(2,9), 685:(1,9), 606:(1,9), 63:(2,9), 705:(4,9), 249:(3,9), 565:(3,9), 293:(1,9), 367:(1,9), 30:(3,9), 554:(4,9), 44:(5,9), 545:(3,9), 318:(1,9), 811:(1,9), 336:(2,9), 913:(2,9), 783:(1,9), 36:(1,9), 938:(1,8), 624:(1,8), 418:(1,8), 472:(3,8), 553:(1,8), 496:(1,8), 143:(1,8), 676:(1,8), 611:(2,8), 102:(2,8), 987:(1,8), 851:(1,8), 658:(2,8), 6:(1,8), 442:(2,8), 193:(1,8), 10:(4,8), 71:(5,8), 533:(1,8), 536:(2,8), 751:(2,8), 932:(1,8), 81:(3,8), 277:(1,8), 280:(2,8), 281:(4,8), 733:(1,8), 91:(1,8), 295:(3,8), 432:(4,8), 711:(1,8), 197:(1,8), 541:(3,8), 82:(2,7), 563:(1,7), 33:(2,7), 585:(3,7), 34:(2,7), 507:(2,7), 713:(1,7), 958:(1,7), 139:(1,7), 832:(1,7), 807:(1,7), 360:(1,7), 399:(2,7), 259:(2,7), 250:(2,7), 934:(1,7), 935:(1,7), 431:(1,7), 842:(2,7), 734:(2,7), 968:(1,7), 720:(2,7), 174:(1,7), 672:(2,7), 147:(3,7), 481:(2,7), 438:(2,6), 323:(1,6), 422:(1,6), 466:(1,6), 402:(3,6), 467:(1,6), 990:(1,6), 352:(1,6), 27:(1,6), 635:(1,6), 556:(1,6), 744:(1,6), 620:(1,6), 675:(2,6), 164:(1,6), 688:(1,6), 227:(3,6), 46:(2,6), 770:(1,6), 232:(1,6), 752:(1,6), 794:(1,6), 780:(2,6), 746:(1,5), 491:(2,5), 996:(1,5), 789:(1,5), 452:(1,5), 722:(2,5), 430:(2,5), 88:(3,5), 956:(1,5), 855:(1,5), 433:(1,5), 230:(1,5)}\nLength of walk: 301. Number of unique nodes in the walk: 171

Walk 3: {823:(3,38), 844:(1,26), 518:(1,25), 45:(2,22), 468:(5,17), 501:(6,16), 185:(2,16), 512:(1,16), 236:(5,16), 627:(1,15), 522:(1,15), 126:(1,15), 629:(2,14), 485:(1,14), 35:(3,14), 94:(1,13), 141:(1,13), 595:(1,13), 745:(1,13), 469:(1,13), 105:(2,12), 198:(2,12), 18:(2,12), 125:(2,12), 513:(2,12), 79:(1,12), 558:(4,12), 631:(1,12), 650:(2,12), 623:(2,12), 177:(3,12), 306:(2,11), 98:(1,11), 64:(2,11), 535:(1,11), 22:(1,11), 127:(1,11), 118:(1,10), 709:(2,10), 515:(3,10), 699:(3,10), 73:(3,10), 525:(3,10), 144:(3,10), 560:(1,10), 42:(3,10), 523:(4,10), 100:(1,10), 110:(1,10), 619:(1,10), 28:(2,10), 12:(2,10), 120:(1,9), 606:(4,9), 648:(1,9), 725:(2,9), 716:(1,9), 637:(4,9), 691:(1,9), 715:(1,9), 657:(4,9), 160:(3,9), 634:(1,9), 685:(1,9), 628:(2,9), 549:(3,9), 36:(3,9), 547:(1,9), 464:(1,9), 455:(1,9), 336:(1,9), 545:(2,9), 913:(1,9), 63:(1,9), 811:(1,9), 318:(1,9), 565:(1,9), 929:(1,8), 143:(1,8), 539:(1,8), 948:(1,8), 938:(1,8), 137:(1,8), 158:(2,8), 533:(1,8), 442:(1,8), 453:(3,8), 454:(2,8), 459:(2,8), 987:(1,8), 2:(3,8), 102:(3,8), 228:(2,8), 571:(1,8), 193:(1,8), 81:(3,8), 733:(1,8), 732:(3,8), 91:(1,8), 751:(2,8), 280:(2,8), 205:(1,8), 295:(1,8), 67:(1,8), 608:(3,8), 564:(1,8), 52:(1,8), 687:(2,8), 553:(1,8), 611:(1,8), 103:(1,7), 576:(1,7), 563:(3,7), 519:(1,7), 104:(2,7), 40:(4,7), 82:(2,7), 507:(5,7), 124:(1,7), 735:(2,7), 986:(1,7), 968:(1,7), 959:(2,7), 958:(1,7), 401:(2,7), 908:(1,7), 906:(4,7), 388:(1,7), 340:(1,7), 138:(2,7), 821:(1,7), 810:(1,7), 792:(1,7), 840:(1,7), 734:(1,7), 210:(1,7), 162:(1,7), 139:(2,7), 147:(2,7), 702:(2,7), 659:(3,7),

Prompt example for number of clusters prediction (continued)

744:(2,6), 163:(1,6), 924:(1,6), 930:(1,6), 875:(3,6), 422:(1,6), 438:(1,6), 145:(1,6), 466:(1,6), 108:(1,6), 990:(1,6), 369:(1,6), 164:(3,6), 856:(1,6), 183:(2,6), 46:(1,6), 770:(1,6), 317:(1,6), 567:(1,6), 302:(1,6), 854:(1,6), 795:(1,6), 730:(1,6), 977:(1,5), 491:(1,5), 479:(1,5), 271:(1,5), 789:(1,5), 955:(3,5), 136:(1,5), 722:(2,5), 919:(2,5), 833:(3,5), 333:(2,5), 894:(1,5)}\nLength of walk: 301. Number of unique nodes in the walk: 176

Walk 4: {311:(6,46), 910:(4,38), 712:(1,20), 642:(3,19), 857:(3,19), 20:(2,17), 296:(1,17), 7:(3,17), 381:(4,16), 244:(1,15), 17:(1,14), 339:(3,14), 909:(2,14), 189:(3,14), 23:(3,14), 636:(3,13), 261:(3,13), 521:(2,13), 671:(1,13), 5:(3,13), 269:(1,13), 692:(1,12), 710:(2,12), 589:(6,12), 584:(1,12), 299:(4,12), 765:(2,12), 37:(2,12), 24:(1,12), 211:(2,12), 256:(1,11), 726:(2,11), 282:(1,11), 180:(3,11), 393:(2,11), 161:(3,11), 31:(3,11), 654:(3,11), 146:(3,11), 666:(4,11), 190:(2,11), 279:(2,11), 167:(4,11), 184:(4,11), 741:(3,10), 240:(1,10), 756:(1,10), 176:(1,10), 255:(3,10), 690:(1,10), 504:(1,10), 803:(2,10), 776:(3,10), 202:(3,10), 133:(6,10), 582:(4,10), 274:(1,10), 192:(1,10), 3:(1,10), 292:(2,10), 364:(1,10), 238:(4,9), 204:(3,9), 241:(1,9), 743:(1,9), 216:(1,9), 425:(1,9), 719:(2,9), 858:(2,9), 420:(2,9), 316:(4,9), 697:(3,9), 366:(1,9), 879:(1,9), 891:(2,9), 243:(2,9), 682:(1,9), 798:(2,9), 310:(1,9), 797:(1,9), 308:(4,9), 812:(1,9), 503:(3,9), 182:(4,8), 156:(1,8), 630:(1,8), 168:(2,8), 483:(2,8), 278:(1,8), 351:(4,8), 152:(1,8), 787:(1,8), 982:(1,8), 949:(1,8), 343:(1,8), 300:(3,8), 775:(2,8), 632:(1,8), 131:(1,8), 647:(1,8), 681:(2,8), 649:(5,8), 291:(1,8), 231:(4,8), 848:(1,8), 737:(1,8), 653:(1,8), 440:(2,7), 149:(2,7), 426:(1,7), 477:(2,7), 157:(1,7), 427:(1,7), 153:(2,7), 700:(1,7), 380:(2,7), 800:(1,7), 328:(3,7), 819:(1,7), 747:(1,7), 224:(2,7), 849:(1,7), 348:(1,7), 287:(3,7), 471:(3,6), 947:(1,6), 784:(2,6), 294:(1,6), 282:(2,6), 487:(2,6), 447:(4,6), 489:(1,6), 313:(1,6), 761:(3,6), 757:(1,6), 324:(1,6), 383:(1,6), 135:(1,6), 922:(2,6), 693:(1,6), 882:(2,6), 409:(1,6), 506:(1,6), 760:(2,5), 845:(1,5), 962:(2,5), 346:(2,5), 349:(1,5), 984:(2,5), 901:(1,5)}\nLength of walk: 301. Number of unique nodes in the walk: 150

Walk 5: {45:(6,22), 468:(2,17), 512:(3,16), 501:(4,16), 785:(4,16), 629:(3,14), 35:(5,14), 485:(5,14), 94:(6,13), 469:(1,13), 650:(1,12), 177:(11,12), 125:(3,12), 513:(6,12), 558:(2,12), 623:(2,12), 105:(2,12), 127:(4,11), 411:(2,11), 64:(6,11), 448:(4,11), 718:(1,11), 799:(4,10), 561:(1,10), 118:(5,10), 699:(1,10), 523:(3,10), 515:(1,10), 144:(3,10), 685:(7,9), 715:(4,9), 716:(1,9), 721:(10,9), 11:(1,9), 160:(3,9), 648:(1,9), 637:(2,9), 63:(2,9), 565:(1,9), 44:(1,9), 39:(2,9), 30:(1,9), 606:(2,9), 705:(2,9), 913:(5,9), 983:(1,9), 318:(2,9), 464:(3,9), 454:(4,8), 453:(3,8), 929:(2,8), 611:(2,8), 608:(2,8), 81:(2,8), 137:(2,8), 143:(1,8), 593:(2,8), 71:(3,8), 154:(1,8), 564:(2,8), 948:(3,8), 197:(4,8), 538:(3,8), 295:(2,8), 10:(2,8), 6:(1,8), 472:(4,8), 2:(1,8), 751:(3,8), 289:(1,7), 968:(2,7), 959:(3,7), 40:(2,7), 585:(2,7), 613:(7,7), 986:(1,7), 792:(1,7), 935:(1,7), 507:(3,7), 138:(3,7), 340:(3,7), 840:(3,7), 659:(1,7), 147:(1,7), 908:(2,7), 399:(3,7), 832:(1,7), 720:(1,7), 734:(2,7), 124:(1,7), 466:(1,6), 862:(1,6), 457:(1,6), 467:(2,6), 730:(1,6), 232:(1,6), 967:(3,6), 556:(1,6), 635:(3,6), 422:(1,6), 438:(2,6), 108:(1,6), 930:(3,6), 402:(1,6), 164:(3,6), 88:(2,5), 436:(2,5), 855:(1,5), 136:(2,5), 955:(3,5), 722:(3,5), 894:(10,5), 452:(1,5), 430:(3,5), 252:(1,5), 814:(3,5)}\nLength of walk: 301. Number of unique nodes in the walk: 116

Walk 6: {823:(5,38), 844:(4,26), 590:(2,26), 703:(1,22), 853:(6,19), 185:(1,16), 236:(2,16), 522:(4,15), 544:(3,15), 627:(3,15), 126:(2,15), 580:(1,14), 404:(1,13), 101:(2,13), 595:(4,13), 141:(5,13), 724:(4,13), 257:(1,13), 745:(1,13), 602:(3,12), 79:(3,12), 511:(1,12), 631:(1,12), 805:(3,12), 652:(3,12), 198:(2,12), 643:(3,12), 89:(2,12), 220:(3,12), 51:(3,12), 830:(1,11), 306:(2,11), 753:(1,11), 557:(3,11), 543:(1,11), 535:(1,11), 480:(2,10), 28:(1,10), 546:(2,10), 529:(2,10), 766:(1,10), 8:(1,10), 551:(3,10), 651:(2,10), 555:(1,10), 847:(1,10), 560:(1,10), 50:(2,10), 110:(1,10), 73:(3,10), 577:(3,10), 605:(4,10), 100:(3,10), 120:(1,9), 691:(1,9), 74:(1,9), 725:(1,9), 575:(3,9), 172:(2,9), 568:(2,9), 587:(4,9), 669:(2,9), 36:(1,9), 618:(3,9), 547:(2,9), 554:(1,9), 634:(3,9), 1:(2,9), 831:(1,9), 455:(2,9), 387:(3,9), 293:(4,9), 249:(1,9), 811:(4,9), 851:(1,8), 347:(1,8), 68:(1,8), 67:(1,8), 93:(2,8), 111:(1,8), 222:(1,8), 281:(1,8), 277:(1,8), 869:(2,8), 78:(4,8), 658:(3,8), 418:(1,8), 155:(1,8), 151:(3,8), 159:(3,8), 446:(2,8), 459:(3,8), 533:(3,8), 205:(1,8), 711:(4,8), 193:(1,8), 228:(2,8), 536:(2,8), 539:(3,8), 932:(2,8), 181:(1,8), 242:(1,8), 676:(1,8), 87:(1,8), 33:(1,7), 429:(2,7), 576:(2,7), 450:(3,7), 562:(1,7), 895:(1,7), 388:(1,7), 902:(1,7), 338:(3,7), 34:(3,7), 842:(2,7), 696:(1,7), 250:(1,7), 210:(1,7), 660:(1,7), 644:(2,7), 821:(1,7), 103:(2,7), 162:(1,7), 807:(1,7), 276:(1,7), 689:(2,6), 417:(1,6), 163:(3,6), 688:(2,6), 237:(1,6), 920:(2,6), 183:(2,6), 27:(1,6), 744:(1,6), 739:(3,6), 212:(3,6), 408:(3,6), 323:(1,6), 85:(1,6), 354:(1,6), 330:(2,6), 319:(1,6), 317:(2,6), 315:(1,6), 352:(1,6), 820:(1,6), 59:(3,6), 369:(2,6), 567:(1,6), 780:(1,6), 956:(1,5), 263:(1,5), 230:(1,5), 266:(1,5), 977:(1,5), 992:(1,5), 218:(2,5)}\nLength of walk: 301. Number of unique nodes in the walk: 157

Prompt example for number of clusters prediction (continued)

Walk 7: {823:(3,38), 910:(1,38), 301:(7,30), 590:(1,26), 260:(6,25), 703:(1,22), 363:(2,22), 596:(6,20), 706:(1,18), 217:(2,17), 284:(1,16), 601:(2,16), 544:(1,15), 537:(3,15), 14:(1,15), 522:(2,15), 868:(2,14), 524:(1,14), 53:(3,13), 257:(1,13), 595:(1,13), 578:(2,13), 101:(1,13), 957:(1,13), 198:(2,12), 89:(2,12), 805:(2,12), 755:(1,12), 220:(1,12), 51:(3,12), 679:(2,12), 621:(1,12), 520:(2,12), 41:(2,12), 527:(2,11), 303:(1,11), 808:(2,11), 32:(5,11), 586:(1,11), 320:(1,11), 254:(3,11), 229:(1,10), 754:(1,10), 555:(1,10), 19:(1,10), 605:(1,10), 779:(1,10), 267:(1,10), 95:(1,10), 793:(2,10), 480:(1,10), 8:(1,10), 740:(2,10), 58:(1,10), 43:(1,10), 21:(2,10), 100:(1,10), 534:(2,10), 26:(1,10), 729:(3,10), 38:(2,10), 651:(2,10), 657:(1,9), 618:(1,9), 293:(1,9), 806:(3,9), 170:(1,9), 664:(2,9), 542:(2,9), 594:(2,9), 249:(1,9), 214:(1,9), 736:(1,9), 495:(1,9), 329:(4,9), 357:(1,9), 981:(4,9), 974:(1,9), 455:(1,9), 859:(1,9), 387:(2,9), 389:(1,9), 864:(1,8), 158:(1,8), 309:(5,8), 684:(2,8), 863:(1,8), 728:(1,8), 881:(1,8), 219:(1,8), 732:(1,8), 347:(1,8), 350:(2,8), 887:(1,8), 129:(2,8), 374:(1,8), 344:(1,8), 96:(2,8), 67:(1,8), 57:(1,8), 932:(2,8), 428:(1,8), 950:(1,8), 531:(1,8), 991:(1,8), 993:(2,8), 345:(1,8), 265:(1,8), 836:(1,8), 763:(1,8), 275:(1,8), 264:(2,8), 312:(1,8), 242:(2,8), 281:(1,8), 488:(1,8), 843:(2,7), 877:(1,7), 834:(3,7), 33:(1,7), 429:(1,7), 946:(1,7), 75:(2,7), 895:(1,7), 48:(3,7), 562:(1,7), 791:(4,7), 398:(1,7), 678:(1,7), 876:(1,7), 314:(1,7), 713:(1,7), 738:(1,7), 225:(1,7), 860:(1,7), 341:(1,7), 768:(3,7), 203:(3,7), 215:(3,7), 979:(2,7), 332:(1,7), 175:(2,7), 686:(1,7), 794:(1,6), 953:(2,6), 824:(1,6), 298:(1,6), 964:(1,6), 484:(1,6), 486:(1,6), 835:(1,6), 410:(1,6), 788:(1,6), 227:(1,6), 117:(2,6), 226:(1,6), 119:(1,6), 588:(6,6), 373:(2,6), 326:(4,6), 764:(1,6), 780:(1,6), 880:(1,6), 852:(1,6), 245:(2,6), 854:(2,6), 237:(1,6), 396:(2,6), 361:(1,5), 997:(1,5), 866:(1,5), 825:(3,5), 912:(1,5), 971:(2,5), 263:(1,5), 890:(5,5), 937:(2,5), 391:(1,5), 424:(2,5), 394:(1,5), 907:(3,5), 413:(1,5), 815:(1,5)}\nLength of walk: 301. Number of unique nodes in the walk: 183

Walk 8: {597:(11,42), 443:(7,17), 439:(4,16), 65:(3,15), 83:(2,14), 77:(2,14), 893:(2,14), 574:(2,13), 552:(3,13), 581:(6,13), 526:(1,13), 92:(4,13), 201:(3,12), 530:(4,12), 517:(5,12), 166:(6,12), 622:(1,11), 638:(1,11), 171:(4,11), 140:(3,11), 134:(2,11), 927:(4,11), 178:(3,11), 390:(2,11), 121:(1,10), 640:(1,10), 113:(5,10), 61:(1,10), 607:(1,10), 4:(2,10), 583:(2,10), 861:(1,10), 559:(6,10), 0:(1,10), 421:(2,10), 435:(2,10), 683:(2,9), 107:(5,9), 928:(1,9), 262:(3,9), 626:(3,9), 403:(2,9), 355:(3,9), 109:(2,9), 106:(7,9), 615:(7,9), 191:(3,9), 76:(2,9), 169:(6,9), 592:(3,9), 999:(2,9), 898:(2,9), 415:(2,9), 926:(4,8), 56:(1,8), 655:(1,8), 570:(1,8), 114:(1,8), 966:(2,8), 115:(3,8), 612:(1,8), 97:(1,8), 90:(1,8), 600:(2,8), 62:(2,8), 47:(2,8), 960:(1,8), 871:(3,8), 395:(2,8), 777:(1,8), 206:(1,8), 16:(2,8), 200:(1,8), 370:(3,7), 372:(2,7), 941:(4,7), 376:(3,7), 889:(1,7), 9:(2,7), 84:(3,7), 441:(3,7), 392:(2,7), 72:(3,7), 874:(1,7), 976:(1,7), 528:(1,7), 29:(1,7), 809:(4,7), 285:(1,7), 707:(1,7), 194:(3,7), 656:(3,7), 569:(2,7), 665:(4,7), 186:(1,7), 407:(4,7), 961:(3,7), 478:(2,6), 449:(2,6), 978:(2,6), 904:(1,6), 591:(1,6), 925:(1,6), 379:(3,6), 365:(1,6), 358:(3,6), 195:(1,6), 173:(1,6), 670:(1,6), 680:(1,6), 940:(1,6), 661:(4,5), 458:(3,5), 434:(2,5), 461:(3,5), 899:(1,5), 492:(1,5), 936:(1,5), 400:(1,5), 965:(2,5), 70:(4,5), 482:(2,5), 897:(1,5), 493:(1,5)}\nLength of walk: 301. Number of unique nodes in the walk: 124

Walk 9: {823:(1,38), 301:(2,30), 844:(3,26), 260:(1,25), 363:(1,22), 45:(1,22), 703:(6,22), 596:(1,20), 853:(1,19), 706:(3,18), 468:(2,17), 501:(1,16), 236:(1,16), 284:(2,16), 601:(1,16), 537:(1,15), 609:(1,15), 208:(2,15), 544:(1,15), 35:(2,14), 524:(3,14), 404:(4,13), 53:(3,13), 94:(2,13), 234:(2,13), 643:(1,12), 558:(2,12), 602:(1,12), 511:(1,12), 207:(2,12), 679:(3,12), 320:(2,11), 827:(2,11), 22:(1,11), 98:(1,11), 254:(1,11), 557:(1,11), 543:(1,11), 268:(1,11), 306:(1,11), 303:(2,11), 599:(3,11), 766:(1,10), 774:(3,10), 778:(1,10), 100:(1,10), 247:(1,10), 754:(1,10), 740:(2,10), 779:(1,10), 729:(1,10), 267:(1,10), 793:(1,10), 709:(1,10), 451:(1,10), 555:(4,10), 561:(1,10), 49:(2,10), 58:(2,10), 534:(1,10), 26:(2,10), 529:(1,10), 28:(1,10), 42:(1,10), 551:(1,10), 523:(1,10), 974:(2,9), 981:(2,9), 657:(1,9), 547:(2,9), 160:(2,9), 132:(1,9), 664:(1,9), 673:(1,9), 913:(2,9), 944:(2,9), 357:(1,9), 318:(3,9), 565:(1,9), 859:(1,9), 587:(2,9), 74:(1,9), 214:(3,9), 389:(1,9), 63:(1,9), 474:(1,9), 170:(1,9), 327:(1,9), 159:(2,8), 733:(2,8), 488:(1,8), 155:(1,8), 158:(1,8), 228:(1,8), 193:(1,8), 242:(2,8), 222:(2,8), 991:(1,8), 749:(1,8), 181:(2,8), 728:(2,8), 219:(1,8), 536:(2,8), 950:(1,8), 446:(1,8), 52:(2,8), 864:(1,8), 350:(1,8), 345:(1,8), 71:(1,8), 67:(1,8), 610:(2,8), 851:(1,8), 116:(1,8), 772:(1,8), 836:(1,8), 334:(1,8), 312:(2,8), 264:(1,8), 81:(1,8), 277:(1,8), 428:(1,8), 280:(1,8), 432:(4,8), 645:(1,8), 659:(1,7), 701:(2,7), 124:(1,7), 165:(1,7), 103:(1,7), 175:(1,7), 174:(2,7), 633:(1,7), 562:(1,7), 576:(1,7), 248:(1,7), 519:(1,7), 905:(1,7), 865:(1,7), 210:(2,7), 872:(1,7), 873:(1,7), 314:(3,7), 821:(1,7), 810:(1,7), 342:(1,7), 386:(1,7), 286:(3,7), 429:(1,7), 215:(1,7), 738:(3,7), 481:(1,7), 225:(1,7), 444:(1,7), 450:(2,7), 408:(1,6), 588:(1,6), 856:(3,6), 352:(2,6), 988:(1,6), 368:(1,6), 878:(1,6),

Prompt example for number of clusters prediction (continued)

```

396:(1,6), 883:(1,6), 373:(1,6), 438:(1,6), 416:(1,6), 854:(3,6), 196:(1,6), 852:(1,6),
838:(2,6), 223:(1,6), 226:(1,6), 227:(1,6), 337:(1,6), 770:(2,6), 290:(1,6), 302:(2,6),
744:(3,6), 119:(1,6), 625:(1,6), 326:(1,6), 117:(2,6), 620:(1,6), 85:(1,6), 391:(1,5), 221:(1,5),
491:(1,5), 727:(1,5), 479:(4,5), 989:(1,5), 218:(1,5), 980:(1,5), 972:(1,5), 975:(1,5),
973:(1,5), 460:(1,5), 956:(2,5), 252:(1,5), 333:(1,5), 375:(1,5), 888:(1,5), 433:(1,5),
230:(1,5), 890:(1,5), 413:(2,5), 839:(1,5), 919:(1,5), 509:(1,5), 951:(1,5)}\nLength of walk:
301. Number of unique nodes in the walk: 210
Walk 10: {311:(5,46), 910:(7,38), 414:(9,25), 712:(3,20), 857:(5,19), 642:(4,19), 296:(1,17),
20:(3,17), 7:(4,17), 381:(1,16), 244:(3,15), 339:(2,14), 189:(2,14), 909:(2,14), 671:(1,13),
261:(2,13), 269:(2,13), 5:(3,13), 636:(1,13), 521:(3,13), 646:(1,13), 668:(1,12), 24:(5,12),
37:(2,12), 765:(1,12), 589:(3,12), 211:(1,12), 122:(1,12), 86:(1,12), 393:(3,11), 180:(2,11),
641:(1,11), 190:(2,11), 279:(1,11), 146:(2,11), 726:(3,11), 782:(4,11), 654:(1,11), 161:(2,11),
698:(1,11), 167:(4,11), 256:(1,11), 582:(2,10), 292:(2,10), 274:(1,10), 694:(5,10), 255:(1,10),
176:(3,10), 756:(2,10), 240:(1,10), 741:(1,10), 133:(3,10), 192:(1,10), 188:(2,10), 690:(1,10),
776:(1,10), 353:(2,10), 364:(2,10), 310:(2,9), 366:(2,9), 758:(2,9), 243:(2,9), 241:(3,9),
238:(1,9), 743:(1,9), 216:(1,9), 719:(3,9), 769:(3,9), 879:(2,9), 308:(4,9), 697:(1,9),
682:(1,9), 420:(4,9), 425:(1,9), 494:(2,9), 503:(1,9), 858:(3,9), 708:(1,9), 812:(2,9),
316:(1,9), 231:(4,8), 681:(2,8), 662:(3,8), 142:(2,8), 649:(1,8), 704:(1,8), 291:(2,8),
131:(1,8), 982:(1,8), 632:(1,8), 321:(1,8), 630:(4,8), 742:(2,8), 775:(2,8), 771:(2,8),
273:(1,8), 483:(2,8), 351:(2,8), 343:(1,8), 841:(1,8), 477:(2,7), 440:(2,7), 153:(2,7),
149:(2,7), 426:(2,7), 846:(1,7), 819:(2,7), 767:(2,7), 700:(3,7), 380:(1,7), 790:(2,7),
886:(1,7), 498:(1,7), 305:(1,7), 328:(1,7), 287:(1,7), 157:(2,7), 239:(2,7), 135:(1,6),
447:(1,6), 947:(1,6), 489:(4,6), 995:(1,6), 471:(1,6), 510:(1,6), 723:(1,6), 786:(1,6),
282:(3,6), 324:(2,6), 784:(1,6), 922:(2,6), 801:(2,6), 371:(5,6), 383:(2,6), 761:(1,6),
693:(1,6), 179:(2,6), 294:(1,6), 921:(1,6), 867:(1,6), 497:(1,5), 13:(1,5), 349:(1,5), 346:(3,5),
508:(1,5), 828:(1,5), 984:(1,5), 283:(2,5), 962:(1,5), 896:(1,5), 900:(1,5), 911:(2,5),
405:(1,5), 677:(1,5), 901:(1,5)}\nLength of walk: 301. Number of unique nodes in the walk: 155
Walk 11: {45:(8,22), 468:(2,17), 512:(6,16), 785:(6,16), 501:(5,16), 629:(3,14), 35:(6,14),
485:(2,14), 469:(2,13), 94:(7,13), 884:(2,13), 650:(1,12), 125:(5,12), 105:(3,12), 513:(2,12),
558:(5,12), 18:(3,12), 177:(3,12), 411:(4,11), 718:(2,11), 98:(2,11), 127:(3,11), 561:(3,10),
523:(4,10), 515:(1,10), 799:(5,10), 42:(3,10), 144:(3,10), 525:(3,10), 118:(2,10), 637:(1,9),
606:(2,9), 464:(3,9), 160:(3,9), 318:(2,9), 63:(2,9), 565:(2,9), 715:(2,9), 11:(3,9), 648:(2,9),
913:(3,9), 30:(1,9), 545:(1,9), 39:(1,9), 197:(5,8), 102:(1,8), 6:(1,8), 617:(1,8), 624:(2,8),
137:(1,8), 10:(4,8), 987:(4,8), 154:(2,8), 2:(1,8), 472:(4,8), 143:(3,8), 538:(4,8), 564:(2,8),
454:(2,8), 453:(2,8), 751:(3,8), 541:(1,8), 81:(1,8), 929:(2,8), 295:(2,8), 593:(1,8), 611:(3,8),
432:(2,8), 948:(1,8), 442:(1,8), 553:(4,8), 613:(2,7), 40:(4,7), 585:(2,7), 123:(3,7), 563:(2,7),
82:(1,7), 604:(3,7), 507:(1,7), 908:(1,7), 406:(4,7), 340:(2,7), 792:(3,7), 935:(1,7), 958:(2,7),
399:(1,7), 735:(2,7), 734:(1,7), 720:(2,7), 702:(1,7), 701:(1,7), 360:(2,7), 986:(2,7),
672:(1,7), 148:(3,7), 147:(1,7), 832:(1,7), 923:(2,6), 550:(1,6), 402:(1,6), 862:(1,6),
930:(2,6), 423:(1,6), 438:(1,6), 466:(2,6), 467:(1,6), 990:(2,6), 967:(2,6), 675:(3,6),
556:(4,6), 635:(1,6), 108:(6,6), 573:(2,6), 164:(1,6), 730:(2,6), 232:(2,6), 955:(1,5),
150:(1,5), 452:(1,5), 252:(2,5), 88:(1,5), 894:(5,5), 430:(2,5), 789:(2,5), 814:(1,5), 833:(1,5),
855:(1,5), 915:(2,5), 136:(1,5)}\nLength of walk: 301. Number of unique nodes in the walk: 129\n"

```

Figure 14: Prompt example for number of clusters prediction

You are a graph theory expert. Following are statistics of simple random walks performed on a large graph. The walks may not cover the whole graph.\nFor each walk, walk statistics are provided as a dictionary: {<node name>:(<number of times node appears in the walk>, <degree of the node>), }.\nUse this information to predict top-20 nodes that have highest Pagerank centrality values in the graph.\nProvide your final answer in the following format, Top Nodes: <list of node names>.\nWalk 1: {1391:(3,25), 1338:(6,18), 680:(1,18), 727:(3,18), 1309:(2,17), 1491:(2,16), 1000:(1,16), 1292:(1,16), 1744:(2,13), 1521:(4,13), 1790:(1,13), 320:(5,12), 1339:(2,12), 306:(1,12), 917:(1,12), 420:(1,12), 1029:(1,12), 1861:(1,12), 1616:(2,12), 491:(5,12), 410:(2,11), 1347:(1,11), 582:(1,11), 700:(1,11), 405:(3,11), 81:(2,11), 1713:(3,11), 1671:(1,10), 1825:(1,10), 78:(1,10), 1721:(2,10), 66:(1,10), 9:(1,10), 1273:(2,10), 1350:(1,10), 313:(2,10), 1268:(3,9), 364:(1,9), 1047:(1,9), 27:(1,9), 501:(3,9), 1106:(1,9), 115:(1,9), 672:(1,9), 1509:(4,9), 695:(3,9), 743:(1,9), 1437:(2,9), 384:(2,9), 319:(2,9), 1856:(1,9), 1535:(1,9), 1381:(3,9), 345:(1,8), 1767:(1,8), 1766:(1,8), 1758:(1,8), 1736:(1,8), 1502:(1,8), 1723:(1,8), 1511:(1,8), 758:(1,8), 133:(3,8), 615:(3,8), 1518:(1,8), 75:(1,8), 1085:(1,8), 41:(1,8), 1893:(2,8), 720:(1,8), 1341:(1,8), 395:(1,8), 1356:(1,8), 1717:(1,7), 465:(2,7), 1890:(1,7), 1383:(2,7), 1520:(1,7), 858:(1,7), 119:(2,7), 767:(1,7), 1877:(1,7), 696:(1,7), 1839:(2,7), 870:(2,7), 1407:(1,7), 1488:(1,7), 762:(1,7), 1844:(1,6), 1016:(1,6), 1964:(1,6), 936:(5,6), 898:(3,6), 281:(3,6), 1529:(6,6), 113:(2,6), 804:(2,6), 755:(1,6), 346:(1,6), 679:(1,6), 855:(3,6), 996:(1,6), 798:(1,6), 1323:(1,6), 1762:(1,6), 1936:(1,6), 1722:(3,6), 995:(1,6), 1847:(3,5), 964:(3,5), 865:(1,5), 332:(1,5), 351:(3,5), 1009:(1,5), 849:(1,5), 1002:(2,5), 775:(1,5)}\nLength of walk: 201. Number of unique nodes in the walk: 117\nWalk 2: {574:(3,37), 1958:(2,20), 45:(9,20), 22:(2,14), 676:(2,14), 728:(2,12), 1079:(1,12), 1224:(3,12), 40:(2,12), 1578:(3,12), 1266:(4,12), 1735:(2,11), 1093:(3,11), 172:(1,11), 53:(1,11), 1441:(2,11), 564:(1,11), 1060:(3,10), 1632:(3,10), 1563:(3,10), 180:(2,10), 192:(2,10), 11:(1,10), 5:(1,10), 1154:(1,9), 262:(2,9), 1094:(2,9), 1151:(2,9), 575:(1,9), 168:(1,9), 1607:(1,9), 714:(2,9), 670:(2,9), 718:(1,9), 1167:(3,9), 221:(1,9), 548:(3,9), 118:(1,9), 1714:(1,9), 71:(2,8), 1644:(3,8), 671:(1,8), 1697:(2,8), 1653:(4,8), 645:(5,8), 138:(1,8), 689:(5,8), 1291:(2,8), 562:(2,8), 48:(1,8), 554:(1,8), 208:(3,8), 1857:(6,8), 1358:(3,8), 228:(5,8), 13:(3,8), 1603:(2,8), 567:(1,7), 300:(1,7), 1625:(2,7), 1139:(3,7), 500:(1,7), 1725:(1,7), 1477:(4,7), 753:(3,7), 1164:(1,7), 1222:(2,7), 212:(4,7), 207:(1,7), 1178:(2,7), 148:(2,7), 1281:(5,6), 232:(1,6), 1290:(1,6), 759:(3,6), 1368:(1,6), 1171:(1,6), 1229:(1,6), 237:(1,6), 854:(1,5), 1888:(5,5), 226:(1,5), 321:(2,5), 1267:(3,5), 1335:(2,5), 756:(1,5), 1329:(2,5), 291:(2,5), 709:(1,5), 257:(3,5), 1686:(2,5), 1777:(1,5)}\nLength of walk: 201. Number of unique nodes in the walk: 92\nWalk 3: {1974:(8,87), 286:(2,54), 1831:(2,28), 1443:(1,22), 1386:(5,22), 1837:(2,20), 433:(3,20), 1510:(2,19), 1254:(1,15), 1504:(3,15), 1122:(2,14), 1778:(1,14), 1205:(3,13), 792:(2,13), 52:(2,13), 1675:(2,13), 427:(1,13), 1606:(1,12), 1201:(2,12), 593:(1,12), 1658:(3,12), 1992:(1,12), 1577:(4,12), 809:(1,12), 184:(3,12), 1828:(1,11), 1457:(3,11), 432:(2,11), 340:(1,11), 429:(1,11), 1204:(1,11), 296:(1,11), 106:(2,11), 1669:(4,10), 315:(3,10), 1007:(1,10), 378:(3,10), 368:(1,10), 137:(1,10), 785:(3,10), 821:(1,10), 486:(3,10), 1056:(1,10), 112:(1,10), 470:(2,10), 1493:(1,10), 90:(2,10), 63:(2,9), 467:(1,9), 1591:(1,9), 55:(1,9), 1417:(1,9), 1460:(2,9), 424:(1,9), 1402:(1,9), 1915:(1,9), 899:(1,9), 1412:(2,9), 363:(2,9), 1816:(1,8), 1175:(2,8), 1252:(3,8), 509:(1,8), 302:(1,8), 833:(1,8), 492:(1,8), 489:(2,8), 1484:(1,8), 457:(1,8), 968:(2,8), 428:(2,8), 1954:(2,8), 1434:(2,8), 1244:(3,8), 1390:(1,8), 1160:(3,8), 348:(2,8), 1878:(1,8), 1163:(1,8), 1250:(1,8), 1604:(1,7), 1568:(1,7), 852:(1,7), 738:(1,7), 1862:(1,7), 487:(2,7), 1506:(2,7), 1763:(1,7), 903:(1,7), 1919:(1,7), 1453:(1,7), 752:(2,7), 1807:(1,7), 808:(2,6), 950:(1,6), 36:(1,6), 51:(3,6), 565:(3,6), 1486:(1,6), 572:(1,6), 1889:(1,6), 439:(1,6), 939:(2,6), 1953:(2,6), 1948:(1,6), 912:(1,6), 152:(2,6), 789:(1,6), 949:(4,5), 958:(1,5), 977:(1,5), 980:(1,5), 243:(4,5), 877:(3,5), 1040:(1,5)}\nLength of walk: 201. Number of unique nodes in the walk: 115\nWalk 4: {1391:(3,25), 680:(2,18), 1338:(2,18), 727:(2,18), 1000:(1,16), 1292:(4,16), 1594:(1,15), 1021:(1,14), 1790:(1,13), 1521:(1,13), 1616:(3,12), 109:(4,12), 917:(2,12), 1319:(2,12), 306:(4,12), 320:(1,12), 1861:(3,12), 1029:(1,12), 491:(3,12), 723:(2,11), 410:(3,11), 1713:(1,11), 92:(3,11), 81:(1,11), 582:(5,11), 1825:(3,10), 1384:(1,10), 1721:(1,10), 1350:(1,10), 1671:(1,10), 134:(3,10), 1273:(5,10), 66:(3,10), 9:(3,10), 1426:(2,10), 1268:(1,9), 1509:(4,9), 726:(3,9), 384:(1,9), 1363:(1,9), 1437:(1,9), 1856:(1,9), 672:(2,9), 1106:(2,9), 27:(2,9), 1474:(1,9), 1511:(1,8), 41:(4,8), 1518:(3,8), 615:(1,8), 1502:(2,8), 1085:(1,8), 100:(1,8), 1341:(3,8), 848:(3,8), 1849:(1,8), 1356:(1,8), 402:(8,8), 1877:(1,7), 1890:(4,7), 967:(2,7), 1379:(1,7), 119:(1,7), 1383:(3,7), 1407:(1,7), 765:(3,7), 557:(1,7), 819:(2,7), 1455:(1,7), 810:(1,7), 396:(6,6), 1936:(1,6), 616:(2,6), 853:(1,6), 113:(2,6), 629:(3,6), 425:(3,6), 1867:(2,6), 463:(1,6), 499:(1,6), 724:(3,6), 936:(4,6), 1822:(1,6), 1762:(1,6),

Prompt example for top-20 Pagerank centrality nodes prediction (continued)

```

755:(1,6), 996:(1,6), 849:(1,5), 1847:(1,5), 775:(2,5), 1005:(2,5), 888:(1,5), 956:(1,5),
332:(5,5), 1850:(1,5), 865:(4,5), 964:(1,5), 971:(1,5)}\nLength of walk: 201. Number of unique
nodes in the walk: 97\nWalk 5: {1694:(5,100), 783:(2,38), 252:(1,37), 655:(6,35), 337:(3,26),
1385:(1,25), 141:(2,24), 1284:(1,22), 1064:(4,21), 1556:(1,21), 682:(3,18), 520:(1,17),
1091:(2,17), 891:(2,17), 189:(2,15), 445:(4,15), 705:(1,14), 1357:(3,14), 1512:(1,14),
515:(1,14), 840:(1,13), 1138:(2,13), 183:(1,12), 441:(1,12), 1894:(2,12), 275:(1,12),
1137:(2,12), 784:(1,12), 576:(1,12), 294:(2,12), 549:(1,12), 1494:(1,11), 298:(1,11), 901:(2,11),
856:(2,11), 681:(1,11), 216:(3,11), 239:(1,11), 1308:(2,11), 1802:(1,10), 652:(3,10), 761:(1,10),
251:(1,10), 1025:(1,10), 871:(1,10), 600:(2,10), 1052:(2,10), 545:(1,10), 1427:(1,10),
1416:(2,10), 589:(1,10), 323:(1,10), 622:(1,10), 613:(2,9), 588:(1,9), 585:(1,9), 1141:(1,9),
1090:(1,9), 1597:(1,9), 563:(1,9), 1072:(2,9), 135:(2,9), 1135:(2,9), 1664:(1,9), 1037:(2,9),
1656:(1,9), 506:(1,9), 1739:(1,9), 1373:(2,9), 985:(2,9), 1475:(1,9), 1387:(1,9), 361:(3,9),
867:(1,9), 715:(1,9), 716:(1,9), 1707:(1,8), 639:(2,8), 1610:(1,8), 1708:(2,8), 1408:(1,8),
598:(2,8), 1848:(1,8), 1845:(1,8), 1215:(2,8), 1651:(1,8), 1143:(1,8), 1710:(2,8), 249:(1,8),
1541:(2,8), 699:(1,8), 121:(1,7), 1605:(2,7), 684:(1,7), 1695:(2,7), 379:(1,7), 746:(1,7),
1691:(1,7), 1764:(1,7), 1177:(1,7), 1044:(2,7), 111:(2,7), 519:(1,7), 1327:(1,7), 811:(2,7),
1429:(2,7), 365:(1,7), 1439:(1,7), 448:(3,7), 1550:(1,6), 1913:(1,6), 919:(1,6), 1346:(2,6),
1237:(1,6), 627:(2,6), 1142:(5,6), 1301:(1,6), 1809:(1,6), 1795:(1,6), 1786:(1,6), 1700:(1,6),
1330:(2,6), 1896:(1,5), 1369:(1,5), 1619:(1,5), 1305:(1,5), 1803:(2,5), 960:(1,5),
1257:(2,5)}\nLength of walk: 201. Number of unique nodes in the walk: 129\n"

```

Figure 15: Prompt example for top-20 Pagerank centrality nodes prediction

```

<think>\nBased on the analysis of the random walks, the number of nodes and edges in the graph
are estimated as follows:\n\n- The combined walk length is 2891 steps, and the number of unique
nodes found is 1665. Using the formula for coverage in a random walk,  $U = N(1 - e^{-L/N})$ ,
where  $U$  is the number of unique nodes and  $L$  is the walk length, solving for  $N$ 
gives approximately 2409 nodes.\n- For the number of edges, the unique edges from Walk 1 (1374)
and Walk 2 (1390) are used with the formula  $U_e = E(1 - e^{-L/E})$ , where  $U_e$  is the
number of unique edges. Solving for  $E$  gives approximately 14280 from Walk 1 and 14630 from
Walk 2, averaging to about 14455 edges.\n- The average degree of the graph is estimated to be
around 12, which is consistent with the visited node average degree of 20.3 after accounting for
the bias towards higher-degree nodes in the random walk.\n\nThus, the estimates are (2409,
14455).\n\nEstimates: (2409, 14455)"

```

Figure 16: Response example from DeepSeek-V3.1 for the task of graph size estimation. Due to the large amount of text, the reasoning portion of the response is not included.

```

<think>\nBased on the analysis of the random walk statistics from the LFR benchmark graph, the
number of clusters or communities is estimated to be 11. This estimation is derived from the
average number of unique nodes visited per walk (approximately 183.64) and the total number of
nodes in the graph (3000, inferred from node labels ranging from 0 to 2999). Using the
relationship between walk length (301 steps), community size, and unique nodes visited, the
average community size is calculated to be about 277, leading to 11 communities.\n\nNumber of
clusters: 11"

```

Figure 17: Response example from DeepSeek-V3.1 for the task of prediction of number of communities. The actual number of communities in the input graph are 10. Due to the large amount of text, the reasoning portion of the response is not included.

```
<think>\nBased on the analysis of the random walks, the graph exhibits characteristics consistent with a scale-free network, specifically the Barabasi-Albert (BA) model. The presence of nodes with very high degrees (e.g., degrees around 150-190) and many nodes with low degrees (e.g., degrees around 5-6) indicates a power-law degree distribution, which is a hallmark of BA graphs. Additionally, the same high-degree nodes (e.g., nodes 0, 15, 13, 10, and 3) appear across multiple walks, suggesting global hubs, which are typical in BA networks due to preferential attachment. The Erdos-Renyi model is ruled out due to the wide degree variance, which is inconsistent with a Poisson distribution. The grid graph is eliminated because degrees are not uniform. The LFR model is less likely because there is no clear evidence of community structure from the walk statistics, and the consistent presence of global hubs aligns more with BA.\n\nGraph Type: BA"
```

Figure 18: Response example from DeepSeek-V3.1 for the task of graph structure prediction. The actual graph-type for the input graph is BA. Due to the large amount of text, the reasoning portion of the response is not included.