Agent-based Substructure Counting under Local Differential Privacy

Yuting Zhang[1,2] **Kai Wang**[3] **Wei Ni**[4] **Ying Zhang**[5] **Wenjie Zhang**[1]
[1]University of New South Wales [2]Data61, CSIRO
[3]Antai College of Economics and Management, Shanghai Jiao Tong University
[4]Edith Cowan University, Perth [5]Zhejiang Gongshang University
{yutingz, zhangw}@cse.unsw.edu.au w.ni@ecu.edu.au w.kai@sjtu.edu.cn
ying.zhang@zjgsu.edu.cn

Abstract

Recent studies have demonstrated the ability of Large Language Models (LLMs) in processing various graph problems. Substructure counting remains challenging in both scalability and accuracy. Incorporating sensitive edge information into the input prompts also introduces significant privacy risks of exposing the private information of user connections in real-world applications. This paper, for the first time, studies substructure counting for LLMs under edge local differential privacy (LDP) in a multi-agent framework. Unlike the `Naive` approach whose estimation relies entirely on overly dense noisy graphs, the proposed `PSC` framework decomposes substructure counting into node-level tasks distributed among node agents, and embeds the knowledge of distributed algorithms and DP frameworks in the curator agent and privacy controller, respectively. Thus, we can leverage the local neighboring information and reasoning capabilities of node agents to improve the estimation accuracy. Extensive experiments on 6 real-world datasets validate the effectiveness of `PSC` framework for substructure counting tasks under ε-edge LDP. Moreover, the non-DP version of `PSC` also demonstrated superior performance over a single LLM on standard substructure counting tasks.

1 Introduction

As data analysis increasingly relies on LLMs instead of complex code, there is a growing demand to analyze graph data via natural language (Ge et al., 2024; Wei et al., 2024). While significant progress has been made on graph-related tasks such as cycle detection, connectivity, and shortest path (Chen et al., 2024; Hu et al., 2024), little attention has been devoted to substructure counting tasks. Substructure counting counts the occurrences of small connected subgraph patterns, such as k-star and triangle, in a given graph (Yin et al., 2025). It has been extensively studied in graph analysis, as counting substructures is crucial for analyzing connection patterns and identifying dense subgraphs (Li et al., 2025; Yin et al., 2024). The accuracy and scalability of substructure counting with LLMs remain unsatisfactory. For example, triangle counting on graphs with 20-30 nodes achieves an accuracy of only 7.21% (Nguyen and Yan, 2024).

Besides accuracy and scalability, privacy is another major challenge in substructure counting. Malicious adversaries can attempt to bypass API restrictions through jailbreaking attacks, thereby gaining unauthorized access to sensitive information in prompts (Priyanshu et al., 2023; Tang et al., 2023). Edges in a graph often encode sensitive information (Imola et al., 2022b; Zhang et al., 2025; Dhulipala et al., 2022), such as communication records in telecommunications graphs, supplier-customer links in business networks, or patient-doctor associations in healthcare systems. Investigating how to leverage the capabilities of LLMs for substructure counting tasks while preserving the privacy of edge information is essential.

Differential privacy (DP) is a framework that mathematically limits privacy risks by ensuring individual data remains indistinguishable in statistical outputs. It is widely regarded as the golden standard for privacy-preserving computation (Dwork et al., 2014). *Edge local differential privacy (LDP)* is a popular DP model that has been widely applied on graphs, supporting tasks, such as private triangle counting, k-core decomposition, and densest subgraph discovery. Consider a scenario where a healthcare analyst interacts with an LLM to investigate transmission patterns of sensitive diseases, such as HIV among a group of patients. This can be modeled as a substructure counting task for LLMs on a disease transmission network, where nodes represent patients and edges represent epidemiological contacts. The transmission patterns can be analyzed by counting triangles and 2-stars in the disease transmission network to determine whether

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 22518–22533

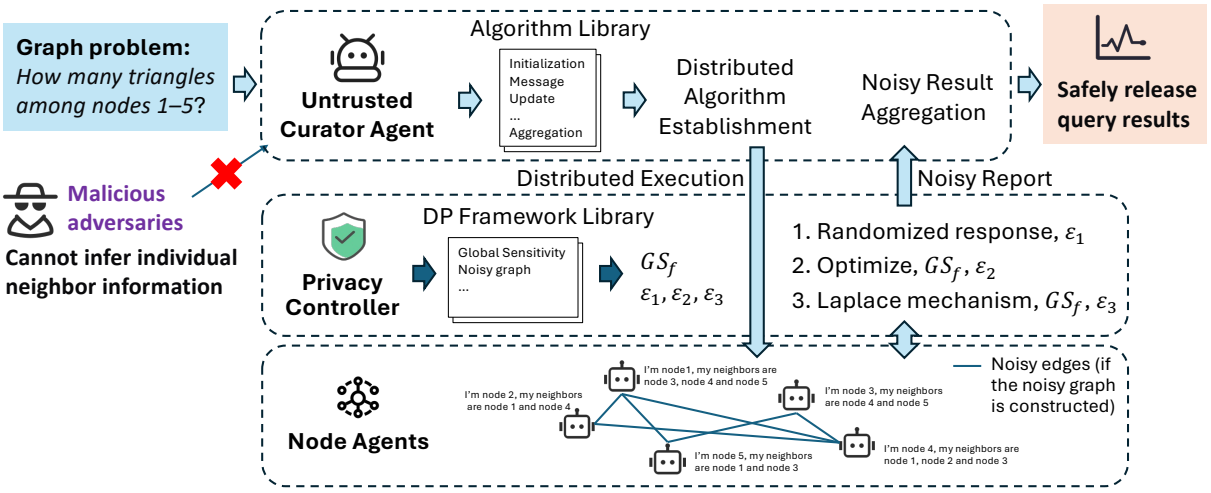


Figure 1: An illustration of the PSC framework. (Given a substructure counting problem, the curator agent establishes an appropriate distributed algorithm for node agents to collaboratively solve the task. Meanwhile, the privacy controller selects a suitable DP framework and determines its parameters. Following the distributed algorithm, each node agent then computes its intermediate counts using its local neighbor information. After execution, node agents report these counts to the curator agent under the protection of the privacy controller. Finally, the curator agent aggregates these reports to produce the final answer.)

the spread is primarily group-based or driven by a super spreader. Without privacy-preserving techniques, responses from LLMs risk exposing patients' confidential relationships. By adopting the LDP framework, the LLM interacts with local agents of each patient's agent. This enables the LLM to answer questions safely without revealing private relationship information.

Motivated by the above discussion, we explore the problem of substructure counting with LLMs under LDP. Given a graph G, a substructure counting task Q, and a privacy budget ε, our goal is to obtain an accurate approximation from LLMs under the protection of ε-edge LDP. In this setting, the LLM that receives the input prompt, called the curator agent, does not have access to edge information and only receives $(V(G), Q)$ as input prompt, where $V(G)$ denotes the node set of the graph G. To produce an estimated count, the curator agent interacts with node agents, each of which manages the neighborhood information for a specific node.

Challenges. We identify the following key challenges to accurately obtain a substructure counting estimation with LLMs under ε-edge LDP.

- Challenge 1: DP requires rigorous mathematically guaranteed protection of individual data. However, LLMs are not well-equipped to reliably execute precise algorithmic and mathematical specifications. Even when an LLM is explicitly instructed to follow DP constraints in the prompt, its outputs may still violate these constraints.
- Challenge 2: One basic approach to estimate the answer under ε-edge LDP is to adopt randomized response (RR) to construct a noisy graph and compute the result on the noisy graph (Imola et al., 2021). However, as shown in previous works (Zhang et al., 2025; He et al., 2024), the generated noisy graph is often much denser than the original graph. The excessive density of the noisy graph often leads to overestimated results. Furthermore, describing the edges of the noisy graph requires a substantially longer input text compared to the original graph. This poses additional challenges for LLMs to produce accurate estimations, as current language models are not robust in leveraging information in long input contexts (Liu et al., 2023).
- Challenge 3: The global sensitivity for some graph reasoning tasks can be very large, leading to injection of excessive noise. For the triangle counting problem, the global sensitivity is $n-2$, as adding or removing an edge in the graph can at most influence $n-2$ triangles, where n is the number of nodes in the original graph. For the k-star, the sensitivity is $\binom{n}{k-1}$, as altering an edge can change at most $\binom{n}{k-1}$ k-stars.

Our approaches. To address Challenge 1, we introduce a privacy controller into the agent-based framework to enforce strict adherence to differential privacy guarantees. The privacy controller selects an appropriate DP framework for the in-

put reasoning task from the DP framework library and determines the required parameters, which are global sensitivity and privacy budget. In this way, each node agent can only communicate with the curator agent and its noisy neighbors with the information perturbed with carefully calibrated noise.

To address Challenge 2, inspired by the two-round framework proposed in prior works (Imola et al., 2021, 2022a), we adopt a distributed execution process that not only enables more accurate estimation but also leverages the reasoning capabilities of the node agents. During this process, node agents interact with their noisy neighbors and the curator agent using perturbed information. We provide aggregate prompts for the curator agent to further enhance the data utility, enabling it to aggregate the answers from the noisy reports effectively.

To address Challenge 3, we integrate the graph projection method into our framework, which removes some neighbors from a neighbor list of the noisy graph so that the maximum degree d_{max} is upper-bounded by a predetermined value $\hat{d}_{max}$, where $\hat{d}_{max} \ll n$. Thus, the global sensitivity of triangle (resp. k-star) counts is at most $\hat{d}_{max}$ (resp. $\binom{\hat{d}_{max}}{k-1}$) after the projection.

Contributions. As the first study of privacy-preserving substructure counting using LLMs under ε-edge LDP, the principal contributions of this paper are summarized below.

- We propose PSC, which enhances data utility by optimizing global sensitivity, utilizing local neighbor information, and leveraging the reasoning capabilities of node agents.
- We conduct extensive experiments on 6 real-world datasets to evaluate the proposed algorithms. Results show that our PSC framework provides better data utility compared to Naive, reducing relative errors by two orders of magnitude on most tests. As detailed in Appendix B, the non-DP version of PSC also improves the accuracy over the state-of-the-art approach for substructure counting without ε-edge LDP.

2 Problem Definition

2.1 Substructure Counting for LLMs

Consider an unweighted and undirected graph $G(V, E)$, where $V(G)$ and $E(G)$ denote the sets of nodes and edges, respectively. Let n and m denote the number of nodes and edges in G, respectively. The adjacency matrix $\mathcal{A}$ for G is of dimensions $n \times n$, where $\mathcal{A}[i, j] = 1$ if there is an edge between v_i and v_j, and 0 otherwise. The i-th row of $\mathcal{A}$ (including both "1"s and "0"s) is the neighbor list of v_i, denoted by $\mathcal{A}_i$. The neighbor set of node v_i is denoted as $N_G(v_i) = \{v_j \in V(G) \,|\, (v_i, v_j) \in E(G)\}$. When discussing LLMs solving graph reasoning problems, the input prompt is formulated as a $(G, \mathcal{Q})$ pair, where the graph G is described as an edge list or adjacency matrices in natural language, and $\mathcal{Q}$ is a task-specific instruction or problem description (Ge et al., 2024; Hu et al., 2024).

2.2 Differential privacy for graph analysis

Classic DP adopts the central model (CDP), where a trusted curator holds all sensitive data and ensures any query on the dataset meets the DP guarantees (Dwork, 2006). This trust assumption is often unrealistic for real-world applications, as a malicious curator could potentially access and leak the entire graph. Moreover, CDP is incompatible with decentralized settings where the graph is distributed across multiple servers. Therefore, we adopt ε-edge LDP, where the curator is untrusted and each user stores its connections in a trusted local server.

Definition 1. **ε-edge LDP.** *Let G be a graph, and $\varepsilon \in \mathbb{R}_{\geq 0}$. For any node $v_i \in V(G)$, let $\mathcal{R}_i$ with domain $\{0,1\}^n$ be a randomized algorithm for node v_i. R_i provides ε-edge LDP if, for any two neighbor lists $\mathcal{A}_i, \mathcal{A}_i' \in \{0,1\}^n$ that differ in one bit and any $\mathcal{S} \subseteq Range(\mathcal{R}_i)$,*

$$\Pr[\mathcal{R}_i(\mathcal{A}_i) \in \mathcal{S}] \leq e^{\varepsilon} \Pr[\mathcal{R}_i(\mathcal{A}_i') \in \mathcal{S}],$$

where ε is the privacy budget.

Note that ε-edge LDP protects a single edge in a neighbor list with privacy budget ε. It guarantees that, given the outputs of the randomized algorithms, two vertices with neighbor lists differing by a single bit cannot be reliably distinguished. Under the setting of ε-edge LDP, the untrusted curator agent receives only the node information of G. Edge information of G is distributed among the node agents, with each node agent $\mathcal{N}_i$ maintaining the neighboring information of node v_i.

Problem Statement. Given an undirected and unweighted graph G, a substructure counting task Q, and a privacy budget ε, we aim to solve the substructure counting tasks for an agent-based framework under ε-edge LDP, where the curator agent takes $P = (V(G), \mathcal{Q})$ as input and returns an answer string R, and each node agent $\mathcal{N}_i \in \mathcal{N}$ holds the neighbor set $N_G(v_i)$ of node v_i. The details of the three substructures are as follows.

T1 Average Degree. The degree of a node is defined as the number of edges connected to it, i.e., $deg_G(v_i) = |N_G(v_i)|$. The average degree of G can be calculated as $\sum_{v_i \in V(G)} deg_G(v_i)/n$.
T2 k-star. A k-star consists of a central node connected to k other nodes. The total number of k-stars in G is $\sum_{v_i \in V(G)} C(deg_G(v_i), k)$.
T3 Triangle. A triangle is a cycle that consists of three nodes and three edges. The number of triangles on G can be calculated as $\frac{1}{3}\sum_{i=1}^{n} c_i$, where c_i denotes the number of triangles including node v_i.

2.3 Differential Privacy Tools

Warner's randomized response (RR) (Warner, 1965). RR is primarily designed to collect truthful responses to sensitive questions while protecting the privacy of respondents in a survey. By applying RR to the neighbor lists, we obtain a noisy graph that satisfies ε-edge LDP (Qin et al., 2017). Given privacy budget ε, a neighbor list $\mathcal{A}_i \in \{0,1\}^n$, each entry $x \in \{0,1\}$ of a neighbor list is flipped with probability $p = \frac{1}{1+e^{\varepsilon}}$. i.e.,

$$RR(x) = \begin{cases} 1-x, & \text{w.p. } \frac{1}{1+e^{\varepsilon}}; \\ x, & \text{w.p. } \frac{e^{\varepsilon}}{1+e^{\varepsilon}}. \end{cases}$$

The state-of-the-art methods on general graphs perturb each entry in the lower triangle of the adjacency matrix to avoid the doubling issue (Imola et al., 2021, 2022a). As proved in (Imola et al., 2021), perturbing each entry in the lower triangle of the adjacency matrix can provide ε-edge LDP.
Calibrating noise to global sensitivity. To achieve ε-edge LDP, any data transmitted from node agents to the curator agent must be perturbed with noise. A widely used approach for adding such noise is the Laplace Mechanism (Dwork et al., 2014), which scales the noise according to the global sensitivity of the data. The concept of global sensitivity is formally defined as follows.

Definition 2. Global sensitivity. *Consider a graph G. Let $\mathcal{A}_i$ be the neighbor list of a node v_i, and $\mathcal{A}'_i$ be a neighbor list that differs from $\mathcal{A}_i$ in one entry. The global sensitivity of a function $f : \mathcal{A}_i \to \mathbb{R}$ is*

$$GS_f = max_{\mathcal{A}_i, \mathcal{A}'_i} |f(\mathcal{A}_i) - f(\mathcal{A}'_i)|.$$

Definition 3. Laplace mechanism. *Given a privacy budget ε and a function f, the Laplace mechanism is*

$$\widetilde{f} = f + \mathrm{Lap}\left(\frac{GS_f}{\varepsilon}\right),$$

where $\widetilde{f}$ is the noisy version of f, and $\mathrm{Lap}(\cdot)$ is the probability density function of the Laplace distribution.

2.4 The `Naive` Approach

One way for the multi-agent framework to estimate the count under ε-edge LDP is to generate a noisy graph G' for the curator agent, allowing it to derive the result on G'. To ensure the ε-edge LDP guarantee, we incorporate privacy controller into the multi-agent framework, perturb the neighbor list of node agents via RR, enabling them to publish the noisy neighbors to the curator agent.

Algorithm 1: `BuildNoisyGraph`

Input: $\mathcal{A}$: an adjacency matrix of graph G; $\mathcal{N}$: node agents; ε: a privacy budget;
Output: G': The noisy graph;
1 **for** *each node agent* $\mathcal{N}_i \in \mathcal{N}$ **do**
2 $\mathcal{A}'_i \leftarrow (RR_\varepsilon(a_{i,1}), ..., RR_\varepsilon(a_{i,i-1}))$
3 **for** *each* $a_{i,j}$ *= 1 with* $j < i$ **do**
4 release the noisy edge (v_i, v_j);
5 curator agent aggregates the noisy graph G';
6 **return** G';

Algorithm 1 outlines the process to build the noisy graph G'. To preserve privacy, each node agent applies the RR mechanism to obfuscate its real neighbor list (Line 2) and reports only the perturbed neighbors with smaller IDs than itself to the curator agent to avoid conflicts (Lines 3-4). The curator agent then merges the edges reported by all node agents to assemble a noisy graph G' (Line 5). The curator agent then encodes the noisy edges of G' as an edge list string and appends it to the original prompt $(V(G), Q)$. The result is then obtained using the updated prompt.

Theorem 1. *The* `Naive` *approach satisfies ε-edge LDP.*

Proof. Algorithm 1 ensures ε-edge LDP by applying the RR mechanism to perturb the original graph using privacy budget ε. Due to the post-processing immunity, any solution derived by the curator agent using the noisy graph continues to satisfy ε-edge LDP. This theorem holds. □

3 The `PSC` Framework

Although the `Naive` approach ensures ε-edge LDP, its estimation relies entirely on the noisy graph. The excessive density of the noisy graph tends to cause overestimation in the results. Furthermore,

the edge information of the noisy graph requires a substantially longer input prompt, which poses additional challenges for the curator agent to produce accurate estimations. Inspired by the two-round framework proposed in prior works (Imola et al., 2021, 2022a), we propose the PSC framework, which incorporates local edge information and leverages the reasoning abilities of node agents. Details of our framework are as follows.

DP parameters determination and noisy node agent network construction. To ensure the ε-edge LDP, the privacy controller first determines the global sensitivity ε for each round, and global sensitivity GS_f according to the input prompt $(V(G), \mathcal{Q})$. Since LLMs lack prior knowledge of the DP framework, we develop a DP framework library for the privacy controller, enabling it to query the relevant framework and determine appropriate parameters. After selecting the DP framework, the privacy controller splits ε into ε_1, ε_2, and ε_3, where ε_1 is used to build the noisy graph, ε_2 is used to optimize the global sensitivity, and ε_3 is used to report the count from each node agent. For algorithms that do not require a specific round, the corresponding privacy budget is set to zero.

Algorithm 2: `GSOptimize`

Input: G': a noisy graph; $\mathcal{N}$: node agents; ε_2: a privacy budget;
Output: GS_f: the optimized global sensitivity; G': an updated G'

1 **foreach** *node agent* $\mathcal{N}_i \in \mathcal{N}$ **do**
2 $\quad \hat{d}_i \leftarrow |\mathcal{N}_i.neighbor| + \mathrm{Lap}\left(\frac{1}{\varepsilon_2}\right)$;
3 $\quad$ report $\hat{d}_i$ to the privacy controller;
4 $\hat{d}_{\max} = \max\left\{\hat{d}_i \mid \mathcal{N}_i \in \mathcal{N}\right\}$;
5 $G' \leftarrow$ `GraphProjection`$(G', \hat{d}_{max})$;
return $GS_f \leftarrow \hat{d}_{max}, G'$;

The global sensitivity for some substructure counting tasks can be large. To mitigate this, we optimize the sensitivity, as summarized in Algorithm 2. Each node agent first reports its degree perturbed with noise sampled from $\mathrm{Lap}\left(\frac{1}{\varepsilon_2}\right)$ to the privacy controller (Lines 1-3). After determining the estimated maximum degree $\hat{d}_{max}$ (Line 4), the privacy controller applies the `GraphProjection` function (Line 5), which ensures that the degree of each node in G' does not exceed $\hat{d}_{max}$. Specifically, for any node whose degree exceeds $\hat{d}_{max}$ in G', only $\hat{d}_{max}$ neighbors are randomly selected and retained. In this way, we restrict the n terms in the global sensitivity expression to $\hat{d}_{max}$.

Algorithm Establishment. To facilitate the curator agent to produce a result under the DP framework, we provide it with a distributed algorithm library. The curator agent specifies the send, update, and aggregate rules for each task. The send rule defines the information that each node agent needs to send to its noisy neighbors. For instance, in triangle counting, each node agent sends its node ID and its noisy neighbor list. The update rule specifies how each node agent counts the required substructures locally. The aggregate rule determines how the curator agent leverages the noisy counts from all node agents to obtain a more accurate count.

Distributed Execution. After the distributed algorithm is designed, each node agent follows the send rules to communicate with its noisy neighbors and then applies the update rules to count the substructure locally. To ensure the differential privacy guarantee, each node agent accesses only the noisy edge memory in the send function. As for the update function, node agents load the real neighbor list. Thus, the updated counting result is kept private and reported to the curator agent under the protection of the Laplace mechanism.

Result Aggregation. Upon completion of the distributed execution, the curator agents determine the final result based on the aggregate rules and the noisy counts from node agents. Details of the aggregation rules are presented in Appendix E.

The procedure of the PSC framework is shown in Algorithm 3. Initially, each node agent $\mathcal{N}_i$ is initialized with the neighbors of v_i in the original graph G, and the count status of each $\mathcal{N}_i$ is initialized as 0 (Lines 1-3). The privacy controller first selects a suitable DP framework to split the privacy budget ε and compute the sensitivity GS_f (Line 4). Then, the noisy graph will be constructed if the algorithm needs a noisy graph (Lines 5-6). If an optimization of the global sensitivity is available, an optimized global sensitivity will be computed (Lines 7-8). Prior to distributed execution, the curator agent designs a distributed algorithm based on the input prompt (Line 9). Subsequently, each node agent constructs and sends messages to its noisy neighbors (Line 10) and updates its count following the instructions (Line 12). After the distributed execution, each node agent will inject noise from $\mathrm{Lap}\left(\frac{GS_f}{\varepsilon_3}\right)$ into its count and send it to the curator agent (Line 13). Curator agent then aggregates all

Algorithm 3: The PSC framework

Input: G: a graph; P: input prompt; ε: a privacy budget; $\mathcal{N}$: node agents;
Output: R: private estimated result;

1 **foreach** $\mathcal{N}_i \in \mathcal{N}$ **do**
2 $\quad \mathcal{N}_i.neighbor \leftarrow N_G(v_i)$;
3 $\quad \mathcal{N}_i.count \leftarrow 0$;
4 determine GS_f and split ε into $\varepsilon_1, \varepsilon_2, \varepsilon_3$;
5 **if** $\varepsilon_1 \neq 0$ **then**
6 $\quad G' \leftarrow$ BuildNoisyGraph$(\mathcal{N}, \varepsilon_1)$;
7 **if** $\varepsilon_2 \neq 0$ **then**
8 $\quad GS_f, G' \leftarrow$ GSOptimize$(G', \mathcal{N}, \varepsilon_2)$;
9 curator agent designs the distributed algorithm according to P;
10 each $\mathcal{N}_i$ sends message M_i to each of its noisy neighbors according to **Send** rules.
11 **foreach** $\mathcal{N}_i \in \mathcal{N}$ **do**
12 $\quad$ update $\mathcal{N}_i.count$ based on the received messages and its own current states by following the **Update** rules;
13 $\mathcal{C} \leftarrow \left\{ \mathcal{N}_i.count + \text{Lap}\left(\frac{GS_f}{\varepsilon_3}\right) \middle| \mathcal{N}_i \in \mathcal{N} \right\}$;
14 curator agent aggregates the final result R according to $\mathcal{C}$ and the **Aggregate** rules;
15 **return** R;

the noisy counts to generate the response (Line 14).

Theorem 2. *Given a graph G and a privacy budget ε, the* PSC *framework satisfies ε-edge LDP.*

Proof. By sequential composition (Dwork et al., 2006), Algorithm 3 satisfies ε-edge LDP with $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$. □

4 Experiments

4.1 Experiments Settings

Algorithms. Our multi-agent framework is built upon AgentScope (Gao et al., 2025), a platform that provides a practical foundation for building scalable, adaptive, and effective agentic applications. We choose GPT-4.1-mini for node agents and GPT-4.1 for the curator agent (OpenAI, 2024). The temperature is consistently set to 0. We evaluate Naive and PSC approaches proposed in Sections 2.4 and 3, respectively. Each algorithm is executed ten times, and the average results are reported. To further assess the performance of LLMs, we compare the results of Naive and PSC with their DP algorithms implemented in Python, denoted as GT-Naive and GT-PSC , respectively. Each Python algorithm is executed 1,000 times, and the average result is taken as the ground truth.

Table 1: Summary of Datasets

Dataset	$\lvert V\rvert$	$\lvert E\rvert$
Contiguous USA (CU)	49	107
Dolphins (Do)	62	159
Les Misérables (ML)	77	254
Ca-sandi_auths (CS)	86	124
Email-enron-only (EE)	143	623
Caenorhabditis elegans (PM)	453	2025

Datasets. We evaluate the performance of degree, 2-star, 3-star, and triangle tasks using 6 real-world datasets in our experiments, where two datasets (EE and CS) are from the Network Repository (https://networkrepository.com), and the remaining four datasets are from KONECT (http://konect.cc/). Table 1 provides the statistics of datasets. $|V|$ (*resp.* $|E|$) denotes the number of nodes (*resp.* edges). To organize the graph information into input prompts, we pre-process the datasets into a unified format as detailed in Appendix A.

Utility metrics. We employ the relative error as the utility metric, following prior works on substructure counting under ε-edge LDP (He et al., 2024; Imola et al., 2022a). The relative error is defined as $\frac{|\hat{f}(G)-f(G)|}{\max\{f(G),\eta\}}$, where $f(G)$ is the true answer of the substructure counting task, $\hat{f}(G)$ is the estimated value of $f(G)$, and $\eta \in \mathbb{R}_{\geq 0}$ is a very small positive value and set as $0.001n$.

Parameters. Following the previous works (Zhang et al., 2025; He et al., 2024), we set the privacy budget ε to 2 by default for all datasets. We allocate $\varepsilon_1 = 0.5\varepsilon$ for noisy graph construction, $\varepsilon_2 = 0.2 \times (\varepsilon - \varepsilon_1)$ for global sensitivity optimization, and $\varepsilon_3 = \varepsilon - \varepsilon_1 - \varepsilon_2$ for the final results. This allocation follows the widely recognized principle that critical information should be allocated a larger portion of the privacy budget compared to auxiliary information (Ye et al., 2020a; Liu et al., 2022). For algorithms that do not require a specific step, the corresponding privacy budget is set to zero while maintaining the proportional allocation of the remaining steps.

4.2 Main Result

Exp1: Performance on all datasets. In this experiment, we evaluate the performance of Naive and PSC on average degree, triangle, 2-star, and 3-star tasks on all datasets. Table 2 shows that, across all datasets and tasks, PSC achieves about

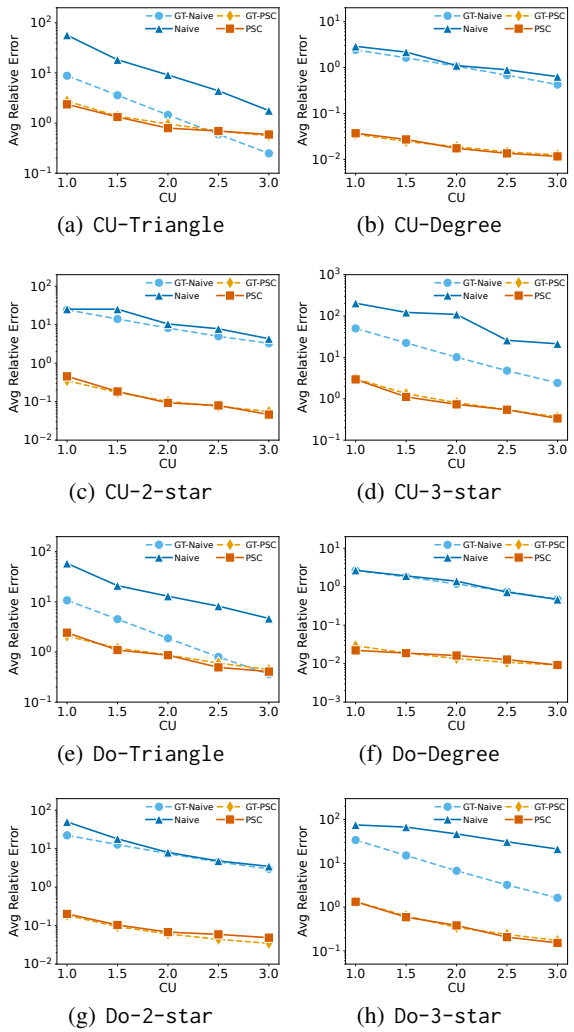
Table 2: Average relative error (↓) of Naive and PSC compare to GT-Naive and GT-PSC

Task	Method	Datasets						Avg.
		CU	Do	ML	CS	EE	PM	
Triangle	GT-Naive	1.4545	1.8761	0.5655	7.0165	1.9941	12.0592	4.1610
	Naive	9.1228	12.9684	7.4197	51.1905	11.4162	32.7652	20.8138
	GT-PSC	0.9612	0.8162	0.3877	1.7108	0.2803	1.0863	0.8737
	PSC	0.7939	0.8628	0.3240	2.9922	0.2914	0.4786	0.9571
Degree	GT-Naive	1.0707	1.1791	1.1351	3.2757	1.7026	5.7853	2.3581
	Naive	1.1020	1.3935	1.8939	4.1415	2.5564	5.3465	2.7390
	GT-PSC	0.0187	0.0143	0.0098	0.0202	0.0052	0.0030	0.0119
	PSC	0.0174	0.0162	0.0110	0.0259	0.0070	0.0666	0.0240
2-star	GT-Naive	3.6030	3.1858	1.9165	13.0441	4.4800	9.9490	6.0298
	Naive	10.4517	7.9931	7.0450	48.4216	24.5390	14.3962	18.8077
	GT-PSC	0.1062	0.0585	0.0444	0.1210	0.0295	0.0284	0.0647
	PSC	0.0929	0.0679	0.0563	0.0800	0.0349	0.3508	0.1138
3-star	GT-Naive	10.1186	6.7480	2.0073	29.8754	7.4755	5.0099	10.2058
	Naive	109.3320	46.9049	9.4129	159.4328	3.1083	33.7694	60.3267
	GT-PSC	0.8035	0.3434	0.2077	0.5410	0.1647	0.1290	0.3649
	PSC	0.7325	0.3816	0.1703	0.4711	0.2445	0.6904	0.4484

one to two orders of magnitude lower average relative error than Naive. Moreover, compared with Naive, the average relative errors of PSC are closer to the relative error of their corresponding ground truth. This demonstrates that using a multi-agent approach for substructure counting is more accurate than relying on a single LLM. Furthermore, the deviation between PSC and GT-PSC is generally smaller for degree and 2-star counting than for triangle and 3-star counting. This is because degree and 2-star counts are easier for smaller LLMs to compute. Regarding graph size, the gap between the average relative error of PSC and the ground truth is smaller on graphs with fewer nodes. This is because nodes tend to have more neighbors in a larger graph, resulting in a lengthy prompt, which is more difficult for the smaller models used by node agents to process accurately.

Exp2: Effect of the privacy budget ε. In this experiment, we report the average relative error of Naive, PSC, as well as their corresponding ground truths on CU and Do datasets, by varying ε from 1.0 to 3.0. The privacy allocation and all other parameters are kept as the default settings. As shown in Fig. 2, the average relative error of all algorithms decreases with increasing ε. This is because a larger privacy budget results in smaller noise scales and thus produces more accurate estimations. Furthermore, the PSC consistently achieves an average relative error up to an order of magnitude lower. For the triangle counting tasks, although the PSC keeps outperforming Naive, when $\varepsilon = 3.0$, the GT-Naive surpasses GT-PSC . This is because when the privacy budget used in building the noisy graph increases, the discrepancy between the noisy graph and the original graph decreases, and this trend is more pronounced for GT-Naive as it allocates the entire ε to generate the noisy graph, while GT-PSC only allocates 0.5ε. The PSC closely tracks the performance of GT-PSC and introduces minimal additional error beyond its inherent error from the DP algorithms. In contrast, Naive deviates significantly from its ground truth, especially for triangle and 3-star counting.

Exp3: Effect of the global sensitivity optimization. In this experiment, we assess the efficiency of the global sensitivity optimization technique for 2-star, 3-star, and triangle counting tasks on two datasets. We denote PSC without global sensitivity optimization as PSC-B and the ground truth of PSC-B as GT-PSC-B. Table 3 shows that PSC and GT-PSC consistently achieve lower relative error compared to PSC-B and GT-PSC-B across all tasks and datasets, respectively. This improvement is particularly notable for the 3-star counting task, as it exhibits higher global sensitivity compared to



(a) CU-Triangle (b) CU-Degree (c) CU-2-star (d) CU-3-star (e) Do-Triangle (f) Do-Degree (g) Do-2-star (h) Do-3-star

Figure 2: Effect of the privacy budget ε

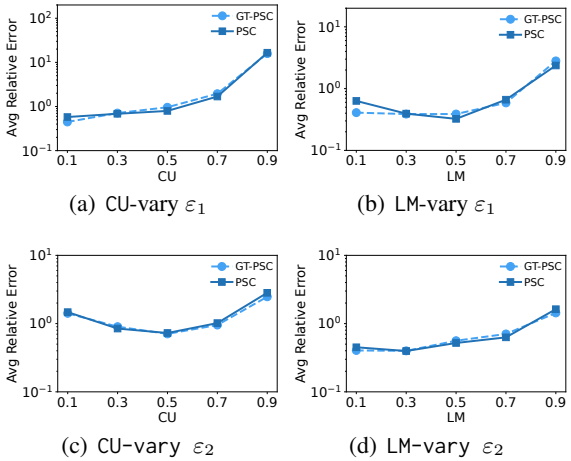
2-star and triangle counting. Furthermore, both PSC-B and PSC show a slight increase in the average relative error compared to the ground truth setting. This is because LLMs, particularly the smaller LLM models used for node agents, may occasionally cause slight errors in the counts.

Exp4: Effect of the privacy budget allocation. In this experiment, we first evaluate the impact of privacy budget allocation between noisy graph construction (ε_1) and result aggregation ($\varepsilon_2 + \varepsilon_3$), while keeping the ratio between ε_2 and ε_3 fixed. We then examine the allocation of the privacy budget within the result aggregation phase by fixing ε_1 and varying the ratio between ε_2 and ε_3. Fig. 3(a) and Fig. 3(b) show the result of varying ε_1 from 0.1ε to 0.9ε and keep $\varepsilon_2 = 0.2(1-\varepsilon_1)$, $\varepsilon_3 = 0.8(1-\varepsilon_2)$. We observe that allocating an excessive portion of the privacy budget to ε_1 significantly increases the average error. This is because, when ε_1 increases, the noisy graph becomes more similar to the original graph. However, if the ε_1 becomes too large,

Table 3: Average relative error (↓) of PSC and PSC without global sensitivity optimization

Task	Dataset	GT-PSC	PSC	GT-PSC-B	PSC-B
Triangle	CU	0.9612	0.7939	1.4714	1.8654
	Do	0.8162	0.8628	1.2985	2.4463
2-star	CU	0.1062	0.0929	0.4487	0.4390
	Do	0.0585	0.0679	0.2580	0.2167
3-star	CU	0.8035	0.7325	4.6370	4.6034
	Do	0.3434	0.3816	2.4373	2.2909

the privacy budget available for reporting the local count becomes insufficient, resulting in excessive noise being injected into the final result and thereby decreasing overall accuracy. In addition, for the CU dataset, the average relative error increases steadily with ε_1, while on the LM dataset, the average relative error initially decreases and then increases. This is because ε_1 is allocated for generating the noisy graph, and the impact of the noisy graph on the final results varies depending on the structure of the specific dataset. Fig. 3(c) and Fig. 3(d) show the results of the privacy budget allocation in the result aggregation phase. We fix the $\varepsilon_1 = 0.5\varepsilon$ and vary ε_2 from $0.1(\varepsilon - \varepsilon_1)$ to $0.9(\varepsilon - \varepsilon_1)$ and kept $\varepsilon_3 = 1 - \varepsilon_1 - \varepsilon_2$. The results show that both very small and very large ε_2 can lead to increased errors.



(a) CU-vary ε_1 (b) LM-vary ε_1 (c) CU-vary ε_2 (d) LM-vary ε_2

Figure 3: Effect of the privacy budget allocation

5 Related work

Edge Local Differential Privacy for Substructure Counting. ε-edge LDP has been widely used in privacy-preserving substructure counting to enhance user privacy (Imola et al., 2021; Ye et al., 2020b; Lv et al., 2021; Eden et al., 2023; Ye et al., 2020a; Hillebrand et al., 2023). Imola et al. (2021) introduce one-round and two-round algorithms for triangle counting and k-star counting under edge

LDP, and further optimize the communication cost and accuracy in (Imola et al., 2022a). He et al. (2025) improve data utility by proposing a vertex-centric method with a further optimized global sensitivity. Hillebrand et al. (2023) optimize the communication cost of subgraph counting under edge LDP via hash functions. Liu et al. (2022) study the triangle counting problem in the localized setting with extended local views. Imola et al. (2022b) estimate 4-cycle and triangle counts under the shuffle model. Liu et al. (2024) enhances data utility for triangle counting under edge LDP through a crypto-assisted approach.

Large Language Models for Graph Reasoning. Empowering LLMs for graph reasoning tasks has attracted increasing research focus in recent studies (Guo et al., 2023; Das et al., 2023; Zhao et al., 2023; Zhou et al., 2023; Luo et al., 2024; Yuan et al., 2024; Perozzi et al., 2024; Tang et al., 2025). Fatemi et al. (2023) explore the impact of different graph encoding methods on graph reasoning tasks with various difficulty levels. Wang et al. (2023) introduce Build-a-Graph and Algorithmic Prompting approaches to improve the LLM performance. Ge et al. (2024) show that the description order of the graph affects the performance of LLMs. Chai et al. (2025) propose GraphLLM, integrating graph learning models with LLMs to boost accuracy. Chen et al. (2024) finetune a language model to solve various types of graph problems and output explicit reasoning paths. Zhang (2023) proposes a graph reasoning interface that enables LLMs to process graph data from various real-world domains. Hu et al. (2024) improve the scalability of graph reasoning tasks by introducing a multi-agent framework. Nguyen and Yan (2024) evaluate the performance with different prompting approaches for the subgraph counting tasks.

6 Conclusion

In this paper, we introduced a multi-agent framework to solve substructure counting tasks under ε-edge LDP. To overcome the issue of excessively long input prompts and overestimation caused by overly dense noisy graphs in the `Naive` approach, we proposed the `PSC` framework that can leverage not only the neighboring information but also the reasoning ability of the node agents. The effectiveness of our proposed techniques is verified through extensive experiments on six real-world datasets.

Limitations

While our `PSC` framework effectively handles the substructure counting tasks under ε-edge LDP, it has several limitations. First, due to the token limitation of LLMs, `PSC` struggles to handle large graphs for triangle counting tasks, which require processing strings including all neighbors. This highlights the need for further improvements in scalability, so that the framework can be effectively applied to graphs with more than 1,000 nodes. We further discuss the potential of scaling `PSC` to larger datasets in Appendix D. Second, the two-round edge LDP algorithms for other graph reasoning tasks are still lacking. Thus, how to effectively extend our framework to solve other graph reasoning tasks under ε-edge LDP remains an open question.

Acknowledgments

Kai Wang is the corresponding author. Kai Wang is supported by NSFC 62302294. Wenjie Zhang is supported by ARC DP260100689 and Australian Research Council Centre of Excellence for Mathematical Modelling of Cellular Systems. We would like to thank the anonymous reviewers for their valuable and constructive feedback.

References

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiang Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. 2025. Graphllm: Boosting graph reasoning ability of large language model. *IEEE Transactions on Big Data*.

Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. 2024. Graphwiz: An instruction-following language model for graph computational problems. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 353–364.

Debarati Das, Ishaan Gupta, Jaideep Srivastava, and Dongyeop Kang. 2023. Which modality should i use–text, motif, or image?: Understanding graphs with large language models. *arXiv preprint arXiv:2311.09862*.

DeepSeek AI. 2024. DeepSeek News API Documentation. Accessed: 9 June 2024.

Laxman Dhulipala, Quanquan C Liu, Sofya Raskhodnikova, Jessica Shi, Julian Shun, and Shangdi Yu. 2022. Differential privacy from locally adjustable graph algorithms: k-core decomposition, low out-degree ordering, and densest subgraphs. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 754–765. IEEE.

Cynthia Dwork. 2006. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 265–284. Springer.

Cynthia Dwork, Aaron Roth, and 1 others. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.

Talya Eden, Quanquan C Liu, Sofya Raskhodnikova, and Adam Smith. 2023. Triangle counting with local edge differential privacy. *arXiv preprint arXiv:2305.02263*.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. *arXiv preprint arXiv:2310.04560*.

Dawei Gao, Zitao Li, Yuexiang Xie, Weirui Kuang, Liuyi Yao, Bingchen Qian, Zhijian Ma, Yue Cui, Haohao Luo, Shen Li, and 1 others. 2025. Agentscope 1.0: A developer-centric framework for building agentic applications. *arXiv preprint arXiv:2508.16279*.

Yuyao Ge, Shenghua Liu, Baolong Bi, Yiwei Wang, Lingrui Mei, Wenjie Feng, Lizhe Chen, and Xueqi Cheng. 2024. Can graph descriptive order affect solving graph problems with llms? *arXiv preprint arXiv:2402.07140*.

Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066*.

Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2024. Common neighborhood estimation over bipartite graphs under local differential privacy. *Proceedings of the ACM on Management of Data*, 2(6):1–26.

Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Wei Ni. 2025. Robust privacy-preserving triangle counting under edge local differential privacy. *Proceedings of the ACM on Management of Data*, 3(3):1–26.

Quentin Hillebrand, Vorapong Suppakitpaisarn, and Tetsuo Shibuya. 2023. Communication cost reduction for subgraph counting under local differential privacy via hash functions. *arXiv preprint arXiv:2312.07055*.

Yuwei Hu, Runlin Lei, Xinyi Huang, Zhewei Wei, and Yongchao Liu. 2024. Scalable and accurate graph reasoning with llm-based multi-agents. *arXiv preprint arXiv:2410.05130*.

Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2021. Locally differentially private analysis of graph statistics. In *30th USENIX security symposium (USENIX Security 21)*, pages 983–1000.

Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2022a. {Communication-Efficient} triangle counting under local differential privacy. In *31st USENIX security symposium (USENIX Security 22)*, pages 537–554.

Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2022b. Differentially private triangle and 4-cycle counting in the shuffle model. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1505–1519.

Shunyang Li, Kai Wang, Wenjie Zhang, Xuemin Lin, and Yizhang He. 2025. Efficient bitruss decomposition on gpu. *IEEE Transactions on Knowledge and Data Engineering*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

Shang Liu, Yang Cao, Takao Murakami, Jinfei Liu, and Masatoshi Yoshikawa. 2024. Cargo: Crypto-assisted differentially private triangle counting without trusted servers. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 1671–1684. IEEE.

Yuhan Liu, Suyun Zhao, Yixuan Liu, Dan Zhao, Hong Chen, and Cuiping Li. 2022. Collecting triangle counts with edge relationship local differential privacy. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2008–2020. IEEE.

Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, and Xing Xie. 2024. Graphinstruct: Empowering large language models with graph understanding and reasoning capability. *arXiv preprint arXiv:2403.04483*.

Tianzi Lv, Huanzhou Li, Zhangguo Tang, Fangzhou Fu, Jian Cao, and Jian Zhang. 2021. Publishing triangle counting histogram in social networks based on differential privacy. *Security and Communication Networks*, 2021(1):7206179.

Ly Nguyen and Yujun Yan. 2024. Evaluating the structural awareness of large language models on graphs: Can they count substructures?

OpenAI. 2024. Gpt-4o: Introducing gpt-4.1 | openai. Accessed: 2024-06-10.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for llms. *arXiv preprint arXiv:2402.05862*.

Aman Priyanshu, Supriti Vijay, Ayush Kumar, Rakshit Naidu, and Fatemehsadat Mireshghallah. 2023. Are chatbots ready for privacy-sensitive applications? an investigation into input regurgitation and prompt-induced sanitization. *arXiv preprint arXiv:2305.15008*.

Zhan Qin, Ting Yu, Yin Yang, Issa Khalil, Xiaokui Xiao, and Kui Ren. 2017. Generating synthetic decentralized social graphs with local differential privacy. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 425–438.

Jianheng Tang, Qifan Zhang, Yuhan Li, and J Grapharena Li. 2025. Benchmarking large language models on graph computational problems. In *International Conference on Learning Representations*.

Xinyu Tang, Richard Shin, Huseyin A Inan, Andre Manoel, Fatemehsadat Mireshghallah, Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, and Robert Sim. 2023. Privacy-preserving in-context learning with differentially private few-shot generation. *arXiv preprint arXiv:2309.11765*.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. 2023. Can language models solve graph problems in natural language? *Advances in Neural Information Processing Systems*, 36:30840–30861.

Stanley L Warner. 1965. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69.

Yanbin Wei, Shuai Fu, Weisen Jiang, Zejian Zhang, Zhixiong Zeng, Qi Wu, James Kwok, and Yu Zhang. 2024. Gita: Graph to visual and textual integration for vision-language graph reasoning. *Advances in Neural Information Processing Systems*, 37:44–72.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. 2020a. Lf-gdpr: A framework for estimating graph metrics with local differential privacy. *IEEE Transactions on Knowledge and Data Engineering*, 34(10):4905–4920.

Qingqing Ye, Haibo Hu, Man Ho Au, Xiaofeng Meng, and Xiaokui Xiao. 2020b. Towards locally differentially private generic graph metric estimation. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1922–1925. IEEE.

Haozhe Yin, Kai Wang, Wenjie Zhang, Yizhang He, Ying Zhang, and Xuemin Lin. 2025. Motif counting in complex networks: A comprehensive survey. *arXiv preprint arXiv:2503.19573*.

Haozhe Yin, Kai Wang, Wenjie Zhang, Ying Zhang, Ruijia Wu, and Xuemin Lin. 2024. Efficient computation of hyper-triangles on hypergraphs. *Proc. VLDB Endow.*, 18(3):729–742.

Zike Yuan, Ming Liu, Hui Wang, and Bing Qin. 2024. Gracore: Benchmarking graph comprehension and complex reasoning in large language models. *arXiv preprint arXiv:2407.02936*.

Jiawei Zhang. 2023. Graph-toolformer: To empower llms with graph reasoning ability via prompt augmented by chatgpt. *arXiv preprint arXiv:2304.11116*.

Yuting Zhang, Wei Ni, Kai Wang, Yizhang He, and Conggai Li. 2025. Truss decomposition under edge local differential privacy. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 2670–2683. IEEE.

Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael Bronstein, Zhaocheng Zhu, and Jian Tang. 2023. Graphtext: Graph reasoning in text space. *arXiv preprint arXiv:2310.01089*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

A Graph Dataset

We pre-process the real-world dataset with the following format:


```
{
  "index": 1,
  "input_prompt": "For each node, the number of 2-stars centered at that node is the number of unordered pairs of its neighbors. This can be calculated as degree(node) * (degree(node) - 1) / 2, where degree(node) is the number of edges connected to the node. The total number of 2-stars in the graph is the sum of 2-stars centered at each node. Each node may hide its edge information under the local edge differential privacy framework. Q: The nodes are numbered from 0 to 48. What is the total number of 2-stars in the graph?",
  "edges": "(0, 1) (0, 2) (0, 3) (0, 4) (1, 2) (2, 4) (2, 30) (2, 31) (3, 4) (3, 5) (3, 6) (4, 5) (4, 7) (4, 26) (4, 30) (4, 41) (5, 6) (5, 7) (5, 8) (5, 9) (6, 9) (7, 8) (7, 17) (7, 18) (7, 32) (7, 33) (7, 41) (8, 9) (8, 12) (8, 16) (8, 17) (9, 12) (10, 11) (10, 12) (10, 13) (10, 14) (11, 13) (11, 15) (12, 16) (13, 14) (13, 15) (13, 37) (14, 16) (14, 19) (14, 37) (15, 37) (15, 39) (16, 17) (16, 18) (16, 19) (17, 18) (18, 19) (18, 32) (18, 35) (19, 35) (19, 37) (19, 38) (20, 21) (20, 22) (20, 23) (21, 22) (21, 23) (21, 45) (21, 46) (22, 28) (22, 29) (22, 46) (24, 25) (24, 26) (25, 26) (25, 27) (25, 29) (25, 44) (26, 30) (26, 41) (26, 44) (27, 28) (27, 29) (28, 29) (29, 43) (29, 44) (30, 31) (32, 33) (32, 34) (32, 35) (32, 36) (33, 36) (33, 40) (33, 41) (34, 35) (34, 36) (34, 48) (35, 38) (35, 48) (36, 42) (37, 38) (37, 39) (38, 48) (40, 41) (40, 42) (40, 43) (41, 43) (41, 44) (42, 43) (43, 44) (45, 46) (45, 47)",
  "vertex_num": 49,
  "edge_num": 107,
  "answer": ###421
}
```


B Additional experimental details

Exp5: Performance of PSC without DP. In this experiment, we evaluate the performance of the PSC framework of substructure counting tasks without the DP protection. As shown in (Nguyen and Yan, 2024), the performance of LLMs in substructure counting tasks can be influenced by different prompting methods. The experiment results indicate that (Few-shot) FS + Least-to-Most (LTM) + 1-hop approach achieves the best performance. To evaluate the performance of the PSC framework on standard substructure counting tasks, we implement a version that removes all injected noise and merges the edge information into the input prompt. We then compare the zero-noise version of PSC with the original prompt (denoted as RAW) and the REFINED prompting method. Further details of the prompts used are as follows.

- **RAW**: Zero-shot prompting provides only the definition of the substructure, along with the task description, including nodes and edge string.

> For each node, the number of 2-stars centered at that node is the number of unordered pairs of its neighbors. This can be calculated as degree(node) * (degree(node) - 1) / 2, where degree(node) is the number of edges connected to the node. The total number of 2-stars in the graph is the sum of 2-stars centered at each node. Q: The nodes are numbered from 0 to 48. The edges are: (0, 1), (0, 2), (0, 3), ... , (45, 47). What is the total number of 2-stars in the graph?

- **REFINED** (Nguyen and Yan, 2024): The best-performed prompt method referred to FS (Few-shot) + Least-to-Most (LTM) + 1-hop in (Nguyen and Yan, 2024), which contains the definition of the substructures, a list of the neighbors of each node, a chain-of-thought reasoning process that breaks down the counting tasks to node-level sub-problems, and a detailed example of the sub-problems using a graph with 10 nodes.

> # The definition of 2-star
> A 2-star graph consists of a central node, called the center, which is connected to exactly two other nodes by edges.
>
> # The COT instruction
> 1. List all edges and identify each node's neighbors.
> 2. For each node, calculate its degree (number of neighbors).
> 3. For nodes with degree >= 2, calculate combinations C(degree, 2).
> 4. Sum all combinations to get the total 2-stars.
>
> # The example with 10 nodes
> This is an example of how you can count 2-star.
> Given a graph has 10 nodes and 38 edges.
> The edge list is: ((3,4), (6,7), (1,2), (2,5), (1,5), (4,7), (9,10), (3,10), (1,4), (3,7), ..., (2,3))
>
> *Step 1*: List neighbors for each node.
> - Node 1: neighbors = [2, 3, 4, 5, 6, 8, 9, 10]
> ...
> - Node 10: neighbors = [1, 2, 3, 4, 5, 6, 7, 9]
>
> *Step 2*: Calculate the degree for each node.
> - Node 1: degree = 8
> ...
> - Node 10: degree = 8
>
> *Step 3*: Calculate $C(degree, 2)$ for nodes with degree >= 2.

Table 4: Average relative error (↓) of RAW, REFINED, and PSC without DP

Task	Method	Datasets						Avg.
		CU	Do	ML	CS	EE	PM	
Triangle	RAW	0.2667	0.6316	3.6304	0.2762	2.2499	2.5283	1.5972
	REFINED	0.2683	0.3193	0.6820	0.3929	0.8121	0.8503	0.5541
	PSC	0.0035	0.0063	0.0128	0.0048	0.0003	0.0094	0.0062
Degree	RAW	0.0056	0.0101	0.1559	0.0685	0.2560	0.2391	0.1225
	REFINED	0.0117	0.0094	0.0177	0.0040	0.1056	1.6781	0.3044
	PSC	0.0011	0.0075	0.0144	0.0013	0.0031	0.0599	0.0145
2-star	RAW	0.0931	0.2414	1.2063	0.2570	0.5427	0.9519	0.5487
	REFINED	0.0000	0.0026	0.0014	0.0059	0.0941	0.4254	0.0882
	PSC	0.0000	0.0000	0.0000	0.0013	0.0173	0.3136	0.0554
3-star	RAW	0.3518	0.4480	1.1728	0.3513	1.0313	1.2114	0.7611
	REFINED	0.0000	0.0099	1.6500	0.0000	0.0445	0.5276	0.0970
	PSC	0.0000	0.0000	0.0000	0.0000	0.0013	0.6838	0.1142

Table 5: Average relative error (↓) of RAW and REFINED prompts for Naive on six datasets

Task	Method	Datasets						Avg.
		CU	Do	LM	CS	EE	PM	
Triangle	Naive +RAW	9.1228	12.9684	7.4197	51.1905	11.4162	32.7652	20.8138
	Naive +REFINED	21.4772	13.0442	27.9884	172.3929	90.0144	119.3614	74.0464
Degree	Naive +RAW	1.1020	1.3935	1.8939	4.1415	2.5564	5.3465	2.7390
	Naive +REFINED	1.3645	1.6490	1.9101	4.2875	1.7669	1.8404	2.1364
2-star	Naive +RAW	10.4517	7.9931	7.0450	48.4216	24.5390	14.3962	18.8077
	Naive +REFINED	4.3050	5.7166	7.7655	20.7292	10.9272	23.6128	12.1760
3-star	Naive +RAW	109.3320	46.9049	9.4129	159.4328	3.1083	33.7694	60.3267
	Naive +REFINED	13.5125	7.7468	15.8053	63.4106	10.2418	0.6741	18.5652

- Node 1: $C(8,2) = 8(8-1)/2 = 87/2 = 28$
...
- Node 10: $C(8,2) = 8(8-1)/2 = 87/2 = 28$

Step 4: Sum all 2-stars.
Total 2-stars $= 28 + ... + 28 = 254$.

The substructure counting task
Follow the given example, you need to answer: What is the total number of 2-stars in the following graph? The edges of this graph are: (0, 1), (0, 2), (0, 3), ..., (45, 47). The nodes are numbered from 0 to 48.

The 1-hop neighbor of the given graph
The one-hop neighboring information of the graph in the problem is:
1-hop neighbor set of node 0 is 1, 2, 3, 4
1-hop neighbor set of node 1 is 0, 2
...
1-hop neighbor set of node 48 is 34, 35, 38

As shown in Table 4, for tasks of triangle, degree, and 2-star, the PSC framework without injected noise achieves the lowest relative error across all datasets. For the prompt methods RAW and REFINED, REFINED outperforms RAW for most cases. However, for simple tasks such as counting average degree, the RAW tends to outperform REFINED. For 3-star counting, the PSC also has the lowest relative error, except for the PM datasets. Because some nodes in the PM dataset have very large degrees, the small LLM cannot accurately count the 3-stars involving those nodes, and it tends to underestimate the actual number.

Exp6: Performance of REFINED on Naive. In the experiment, we evaluate the performance of the best-performing prompting approach in Exp5 when applied to our Naive approach. Table 5 shows that

Table 6: Average relative error (↓) and absolute difference Δ (↓) of other LLMs

Tasks	Dataset	GPT	Δ_{GPT}	DeepSeek	$\Delta_{DeepSeek}$	Qwen	Δ_{Qwen}
Triangle	CU	0.7939	0.1673	0.9766	0.0154	1.0224	0.0612
	Do	0.8628	0.0466	0.6653	0.1509	1.5519	0.7357
Degree	CU	0.0174	0.0013	0.0762	0.0575	0.0199	0.0012
	Do	0.0162	0.0019	0.0214	0.0071	0.0212	0.0069
2-star	CU	0.0929	0.0133	0.2252	0.1190	0.0814	0.0248
	Do	0.0679	0.0094	0.0740	0.0155	0.0547	0.0038
3-star	CU	0.7325	0.0710	0.4589	0.3446	0.1207	0.6828
	Do	0.3816	0.0382	0.4072	0.0638	0.2645	0.0789
Avg. Δ		–	0.0436	–	0.0968	–	0.1994

REFINED performs well on degree counting for large datasets. For the 3-star and 2-star counting tasks, REFINED outperforms RAW dramatically on some datasets, while RAW remains better on others. However, on triangle counting tasks, REFINED consistently performs worse than RAW. Compared with Exp5, where REFINED outperforms RAW on all datasets for 2-star counting and most of the datasets for 3-star counting and triangle counting, REFINED is unsuitable for improving the performance of a single LLM when processing noisy graphs.

Exp7: Performance of PSC using other LLMs. In this experiment, we evaluate DeepSeek-Chat-V3.1 (DeepSeek AI, 2024) for both node and curator agents, as well as Qwen3-8B for node agents and Qwen-plus for curator agents (Yang et al., 2025). To quantify the error introduced by each model, we measure the absolute difference in Relative Error (RE) between the result of LLMs and the ground-truth from LDP algorithms, defined as $\Delta = |RE_{\text{LLM}} - RE_{\text{GT}}|$. The results in Table 6 demonstrate that our framework is compatible with various LLMs and also works effectively with smaller models such as Qwen3-8B. In terms of absolute difference, GPT consistently exhibits low deviation across all tasks. Furthermore, DeepSeek-Chat-V3.1 achieves accuracy comparable to the ChatGPT models. However, Qwen3-8B and Qwen-plus display much larger deviations, particularly in 3-star and triangle counting. The greater deviations are attributable to the use of smaller models for node agents (e.g., Qwen3-8B), which tend to introduce higher inaccuracies, especially for more complex graph statistics.

C Cost Analysis

In this subsection, we analyze the time and communication cost of the algorithms used in the PSC framework. Let m and m' denote the number of edges in the original graph G and noisy graph G', respectively, $deg_G(v_i)$ and $deg_{G'}(v_i)$ denote the number of neighbors of v_i in the original graph G and noisy graph G', and n denotes the number of nodes in G and G'.

Average Degree. The time complexity for each node v_i to count its neighbors is $O(1)$. Thus, count the average degree takes $O(n)$ time in total. For the communication cost, as it requires each node to report its noisy degree, the communication complexity is $O(n)$.

k-star. For the first step, estimating the maximum noisy degree to optimize the global sensitivity requires $O(n)$ time. Each node takes $O(1)$ time to compute the number of k-stars. Thus, the k-star takes $O(n)$ time in total. For the communication cost, as it requires each node to report its noisy degree and noisy k-star count, the cost is $O(n)$.

Triangle. In the first round, generating the noisy graph takes $O(\binom{n}{2})$ time. Estimating the maximum noisy degree takes $O(n)$ time. Counting the number of triangles for each node requires each node to count triangles in its local graph, which needs $O(\sum_{v_i \in V(G)} deg_G(v_i)^2)$. Thus the time complexity is $O(n^2 + \sum_{v_i \in V(G)} deg_G(v_i)^2)$ in total. For the communication, as it requires each node to obtain the noisy neighbors of its noisy neighbors, the domain communication cost is $O(\sum_{v_i \in V(G)} deg_{G'}(v_i)^2)$. Here, $\mathbb{E}(deg_{G'}(v_i)) = O\left(\frac{n + deg_G(v_i)(e^{\varepsilon_1}-1)-1}{1+e^{\varepsilon_1}}\right)$.

Exp8: Token consumption of G and G'. In this experiment, we demonstrate the token consumption to describe the original graph G and the noisy graph G'. The results are shown in Table 7. Since the expected edge number of the noisy graph G' is $\mathbb{E}(m') = O\left(\frac{\binom{n}{2}+m(e^{\varepsilon_1}-1)}{1+e^{\varepsilon_1}}\right)$, the token consumption of describing the edges of the noisy graph is relatively higher.

Table 7: Token counts for original graphs and noisy graphs

Dataset	G	G'
CU	1.6K	5.6K
Do	2.2K	7.2K
LM	3.3K	9.5K
CS	1.8K	9.5K
EE	7.8K	22.7K
PM	24.6K	158.5K

D Scale PSC to larger datasets

The current scalability limitation is caused by the context window limitation of a single LLM. As LLMs become capable of supporting larger context windows, PSC can scale to much larger graphs without structural changes. Besides, graph sampling enables our framework to handle larger graphs by reducing the input size to fit within the LLM's context window.

Semantics-based sampling. In practical applications of graph-related queries, graphs can be filtered into smaller, more focused subgraphs based on query conditions (e.g., attributes, timestamps, or query nodes), which enables PSC to be applied to larger graphs.

Structure-based sampling. Substructure counts can be accurately estimated using well-established sampling techniques.

Table 8: Average relative error (↓) of PSC on sampled graph G'.

Method	Facebook	Bible
GT-PSC on G	0.8226	0.6966
PSC on sampled G'	0.7722	1.4386

Exp9: Performance of node sampling on larger datasets. We conducted an experiment on Facebook ($n = 4,039$, $m = 88,234$) and Bible ($n = 1,773$, $m = 9,131$) to demonstrate the performance of PSC on larger graphs using random node sampling. In this experiment, we randomly sample 200 nodes to form a subgraph and estimate the total triangle count as $|\hat{\triangle}| = |\triangle|' \times \frac{\binom{n}{3}}{\binom{n'}{3}}$, where n' and $|\triangle|'$ are the number of nodes and triangles in the sampled subgraph. The results are shown in Table 8, which demonstrates that, when combined with sampling, PSC can effectively estimate substructure counts on large graphs, allowing for a flexible trade-off between accuracy and scalability.

E The Library of PSC

An example of distributed algorithm in the algorithm library.

Triangle Counting

Distributed Algorithm for Triangle Counting
Problem: Count the total number of triangles in a given graph G (a triangle consists of three nodes, each connected to the other two).

State:
1. Triangle_Count: The number of triangles this node participates in.
2. Triangles: A list of detected triangles, each represented as an unordered set.

Message:
1. 'Sender_ID': The sender node's ID.
2. 'Sender_Neighbors': The list of the sender's neighbors.

Initialization
Step 1: Set 'Triangle_Count' to 0.

Send:
For each node, the input is:
Node ID: <node_id>
State: 1. 'Triangle_Count': <triangle_count>
Neighbor Information:
Connected to:
Node A, Node B
Step 1: For each neighbor, construct the message:
1. 'Sender_ID': <node_id>
2. 'Sender_Neighbors': <neighbor_list>

Example
Input:
Node ID: 5
State: 1. 'Triangle_Count': 0
Neighbor Information:
Connected to:
Node A, Node B
Process:
For Node A,
Step 1: Construct the message to Node A: 1. 'Sender_ID': 5; 2. 'Sender_Neighbors': [A, B]
For Node B,
Step 1: Construct the message to Node B: 1. 'Sender_ID': 5; 2. 'Sender_Neighbors': [A, B]
Output:
Message sent to Node A, B: 1. 'Sender_ID': 5; 2. 'Sender_Neighbors': [A, B]

Update:
For each node, the input is:
Node ID: <node_id>
State: 1. 'Triangle_Count': <triangle_count>
Neighbor Information:
Connected to:
Node A, Node B
Received Messages:
Message: 1. 'Sender_ID': <sender_id>; 2. 'Sender_Neighbors': <sender_neighbor_list>
Message: 1. 'Sender_ID': <sender_id>; 2. 'Sender_Neighbors': <sender_neighbor_list>

Process
Step 1: For each received message, extract 'Sender_ID'. If 'Sender_ID' is not in <neighbor_list>, discard the message; otherwise, proceed.
Step 2: Extract 'Sender_Neighbors' from the message. Identify the common neighbors between yourself and 'Sender_ID' by finding which nodes in 'Sender_Neighbors' also appear in <neighbor_list>.
Step 3: For each common neighbor 'C', increment 'Triangle_Count' by 0.5, because a triangle will be counted twice.

Example
Input
Node ID: 5
State: 1. 'Triangle_Count': 0
Neighbor Information:
Connected to:
2 4
Received Messages:
Message: 1. 'Sender_ID': 1; 'Sender_Neighbors': [4, 5]
Message: 1. 'Sender_ID': 4; 'Sender_Neighbors': [2, 5, 8]
Process
From the message from Node 1:
Step 1: 1 is not in <neighbor_list>; discard the message.
From the message from Node 4:
Step 1: 4 is in <neighbor_list>; proceed.
Step 2: Check common neighbors; the common neighbors are [2].
Step 3: For the common neighbor 2, increase 'Triangle_Count' by 0.5.
Output
State: 1. 'Triangle_Count': 1

Aggregation: To determine the final result, sum the results of the distributed execution, then divide this value by 3. If the calculated result is negative, return 0 instead. Compute and return the final result as a single number only.

An example of DP framework in the DP framework library.

Triangle Counting

These are the parameters you need for computing triangles of a given graph under edge local differential privacy.
Sensitivity: degree-1
Noisy_Graph: yes