Practical and Accurate Local Edge Differentially Private Graph Algorithms

Pranay Mundra
Yale University
New Haven, CT, USA
pranay.mundra@yale.edu

Charalampos Papamanthou
Yale University
New Haven, CT, USA
charalampos.papamanthou@yale.edu

Julian Shun
MIT CSAIL
Cambridge, MA, USA
jshun@mit.edu

Quanquan C. Liu
Yale University
New Haven, CT, USA
quanquan.liu@yale.edu

ABSTRACT

The rise of massive networks across diverse domains necessitates sophisticated graph analytics, often involving sensitive data and raising privacy concerns. This paper addresses these challenges using *local differential privacy (LDP)*, which enforces privacy at the individual level, where *no third-party entity is trusted*, unlike centralized models that assume a trusted curator.

We introduce novel LDP algorithms for two fundamental graph statistics: k-core decomposition and triangle counting. Our approach leverages previously unexplored input-dependent private graph properties, specifically the degeneracy and maximum degree of the graph, to improve theoretical utility. Unlike prior methods, our error bounds are determined by the maximum degree rather than the total number of edges, resulting in significantly tighter theoretical guarantees. For triangle counting, we improve upon the previous work of Imola, Murakami, and Chaudhury [43, 44], which bounds error in terms of the total number of edges. Instead, our algorithm achieves error bounds based on the graph's degeneracy by leveraging a differentially private out-degree orientation, a refined variant of Eden et al.'s randomized response technique [27], and a novel, intricate analysis, yielding improved theoretical guarantees over prior state-of-the-art.

Beyond theoretical improvements, we are the first to evaluate the practicality of local DP algorithms in a distributed simulation environment, unlike previous works that tested on a single processor. Our experiments on real-world datasets demonstrate substantial accuracy improvements, with our k-core decomposition achieving errors within **3x** the exact values—far outperforming the **131x** error in the baseline of Dhulipala et al. [19] . Additionally, our triangle counting algorithm reduces multiplicative approximation errors by up to **six orders of magnitude** compared to prior methods, all while maintaining competitive runtime performance.

Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://doi.org/10.5281/zenodo.15741879.

1 INTRODUCTION

Graph statistics such as the k-core decomposition and triangle count provide important characteristics about the underlying graph, such as its well-connected communities. These analytics, often performed on sensitive and private graphs, are regularly shared with a wide audience, including researchers, companies, governments, and the public. As such, it is vital to investigate techniques that can safeguard these published graph statistics from privacy attacks.

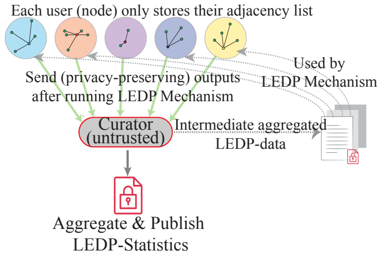

Each user (node) only stores their adjacency list
Send (privacy-preserving) outputs after running LEDP Mechanism
Used by LEDP Mechanism
Curator (untrusted)
Intermediate aggregated LEDP-data
Aggregate & Publish LEDP-Statistics

Figure 1: Local edge differential privacy (LEDP) Model

The *k-core decomposition* assigns an "importance" value to each node, roughly representing its influence within the graph. It is widely used to analyze the structure of real-world graphs, including social, email, and disease transmission networks. Formally, a k-core of a graph is a maximal subgraph where the induced degree of every vertex in the subgraph is at least k. The k-core decomposition (see Definition 2.16 and Figure 2) assigns a number, denoted as $core(v)$, to each vertex v. This number, $core(v)$, represents the largest value of k for which the k-core still includes vertex v. Unfortunately, these values pose privacy risks.

Consider the application of k-core decomposition to COVID transmission data [14, 35, 72, 76, 80] and other disease transmission networks [15] such as HIV [32, 41]. The core numbers are generated and published, sometimes even with location data [79]. Revealing the precise core numbers of every individual can lead to privacy breaches. Consider a scenario where exactly c individuals have a core number of $c - 1$. This implies they form a clique of c vertices, all connected. In disease transmission graphs, this directly exposes a cluster of sensitive disease transmissions! Therefore, it's essential to release privacy-preserving core numbers.

Similarly, *triangle counting* is widely used in applications that process sensitive data. The triangle count, which measures the number of three-node cycles in a graph, is a fundamental metric in community detection [62, 70, 71, 82], recommendation systems [60, 98], and clustering [89]. In databases, triangle counting is essential for graph analytics frameworks [67] and is leveraged in query optimization [4, 8] and fraud detection [85]. However, exposing triangle counts without privacy guarantees can lead to inference attacks that compromise user confidentiality. Recent works in security and privacy [43, 44, 56, 58, 59] have highlighted the risks associated with publishing triangle counts, reinforcing the importance of privacy-preserving graph analytics.

Approximate values for such statistics are widely used to improve scalability and efficiency with minimal utility impact. In graph databases and uncertain networks, approximate cores enable analysis under probabilistic edges [10]; in social and recommendation systems, they support influence detection and improve accuracy [3, 48]. Often, approximate core numbers are used for preprocessing other algorithms such as clustering [33, 63]. Triangle counts are used in clustering, fraud detection, and query optimization, where small errors are tolerable [5, 9, 56]. In dynamic domains like cybersecurity and biology, approximations allow timely insights from noisy data [54, 83, 93]. Approximate statistics with strong privacy guarantees are often practical and effective when these applications use sensitive data [19, 27, 43, 44, 54, 56].

Differential privacy (DP) [23] is often considered the "gold standard" in protecting individual privacy. Traditionally, DP has been studied in the **central model**, where a trusted curator has access to raw data and applies DP mechanisms before releasing the results. However, this assumption is often impractical, especially in modern systems that rely on decentralized or federated architectures. This motivates the **local model** of differential privacy, introduced by the seminal works of Evfimievski et al. [28] and Kasiviswanathan et al. [46], which recently gained much attention in the theoretical computer science [19, 20, 27, 37], cryptography and security [31, 43, 44, 52, 56, 57, 90, 99], data mining [11, 39, 40, 58, 59, 65, 68], query answering [30, 49, 75, 88, 91], and machine learning [34, 36, 38, 45, 53, 64, 78, 92, 95, 100] communities. In this model, each user independently applies DP mechanisms before sharing their privacy-preserving outputs with an untrusted curator. It offers stronger privacy guarantees by never exposing raw data and is naturally suited to distributed settings; it has been used in prominent cases including federated learning [47, 61, 66], the 2020 U.S. Census [2], and by companies such as Apple [86].

While DP in the traditional database setting focuses on protecting individual records, many real-world datasets are inherently relational, represented as graphs. In these cases, the sensitive information are the edges, i.e., the connections between entities. This motivates the need for *local differentially private (LDP) graph algorithms*. The *local edge differential privacy* model (LEDP), as introduced in recent works [19, 43, 73], represents a novel approach designed to ensure local privacy for *graph outputs* (Fig. 1). Graph data is increasingly integral to modern database systems, underpinning applications in knowledge graphs, social networks, cybersecurity, and financial fraud detection. Many relational databases now support graph extensions (e.g., SQL-based graph queries), while specialized graph databases [7, 67] are widely deployed in industry. However, applying LEDP algorithms in these settings is challenging: unlike in tabular data, where individual data points can be perturbed, perturbing edges within a graph introduces structural dependencies and high computational cost.

All previous implementations of local differentially private graph algorithms [42, 44, 73] use *Randomized Response (RR)* [94], which independently flips the presence or absence of each edge with a probability based on the privacy parameter ε. While simple and composable, RR introduces substantial noise, especially for small $\varepsilon \in (0, 1]$, increasing the *density*[1] of the input graph and limiting both the accuracy and scalability of the algorithm. Dhulipala et al. [19] proposed the first LEDP algorithm that goes beyond RR, leveraging the geometric mechanism for k-core decomposition. However, their algorithm is purely theoretical, and their error bounds scale poorly with graph size. In particular, they give additive error bounds $\geq \frac{\log_2^3(n)}{\varepsilon}$ (where n is the number of vertices); on a graph with $n = 10^5$ nodes and $\varepsilon = 0.5$, this translates to an additive error of 9164. Most real-world graphs of this size have max core numbers of 10^2 magnitude, so the additive error itself leads to a ≥ 91-multiplicative approximation factor—much too large for any practical use.

This work simultaneously develops *both* new theoretical and implementation techniques that, together with Randomized Response, enable provably private, accurate, and computationally efficient LEDP algorithms. We make the following contributions:

- We design a *novel* LEDP k-core decomposition algorithm that doesn't use Randomized Response and provides provable privacy and error guarantees. Leveraging the input-dependent maximum degree property of the graph, we achieve improved theoretical bounds over the LEDP k-core decomposition algorithm of Dhulipala et al. [19] (see Table 1). Two key innovations lie in thresholding the maximum number of levels a node can move up based on its noisy (private) degree and the use of bias factors to reduce the impact of noise. Since a node's core number is upper bounded by its degree, our algorithm offers stronger theoretical guarantees for most real-world graphs, where the maximum degree is significantly smaller than the number of nodes.
- We present the *first* implementation of a private k-core decomposition algorithm and demonstrate through empirical evaluation that it achieves an average error of **3x** the exact values, markedly improving upon prior approaches. Furthermore, our LEDP implementation attains error rates that closely align with the theoretical approximation bounds of the best *non-private* algorithms, underscoring its practical efficiency and accuracy.
- We design a *novel* LEDP triangle counting algorithm that modifies our k-core decomposition to construct a low out-degree ordering, minimizing each node's out-degree. Leveraging this ordering, our algorithm achieves improved theoretical error bounds over the best-known methods of Imola et al. [43, 44] and Eden et al. [27] for bounded degeneracy graphs, common in real-world networks (see Table 1). We present a novel analysis to analyze the non-trivial error bounds based on the Law of Total Expectation/-Variance. Our implementation reduces relative error by up to **89x** and improves the multiplicative approximation by up to **six orders of magnitude** over the best previous implementation [44], while maintaining competitive runtime.
- Recognizing that the LEDP model (see Section 2.3) is inherently decentralized, we present the first evaluation of LEDP graph algorithms in a simulated distributed environment with actual message passing. Unlike prior studies that relied on a single processor, we simulate a distributed environment by partitioning the graph across multiple processors. This approach provides a more realistic assessment of both computational and communication overhead in large-scale distributed scenarios. We demonstrate

[1] The density is the ratio of the number of edges to the number of nodes.

	Previous Work	This Work
k-Core	$O\left(\frac{\log^3(n)}{\varepsilon}\right)$ [19]	$O\left(\frac{\log(D_{\max})\log^2(n)}{\epsilon}\right)$
Triangle Counting	$O\left(\frac{n^{3/2}}{\varepsilon^3}+\frac{\sqrt{C_4}}{\varepsilon^2}\right)$ [27]	$O\left(\frac{\sqrt{n}d\log^3 n}{\varepsilon^2}+\sqrt{\overrightarrow{C}_4}\right)$

Table 1: Additive error bounds compared to previous work. Here n : number of nodes, D_{max} : maximum degree, ε : privacy parameter, d : degeneracy, and C_4 and $\overrightarrow{C}_4$ is the number of 4-cycles and oriented 4-cycles, respectively.

the practicality of this evaluation by applying it to our k-core decomposition and triangle counting algorithms.

- We present the first LEDP graph algorithm implementations that scale to ***billion***-edge graphs, whereas prior implementations were tested on graphs with millions of edges [43, 44]. Our evaluation framework serves as a valuable tool for designing and testing other LEDP algorithms. Our source code is available at [1].

1.1 Related Work

Local differential privacy (LDP) for graph data has been extensively studied [19, 27, 39, 43, 44, 73, 84, 96, 97], focusing on tasks such as synthetic graph generation and subgraph counting. Some works [58, 84] explore an *extended local view*, in which each node knows its full 2-hop neighborhood (i.e., its neighbors' edges) to improve triangle counting accuracy. In that model, triangle counting is trivial since each node sees its entire set of incident triangles—unlike our model, where nodes see only immediate (one-hop) neighbors; hence, we require more complex algorithms since nodes cannot see their incident triangles in LEDP. Moreover, the extended view is often unrealistic, since users (e.g., in social networks) may not wish to reveal their private (potentially sensitive) friend lists to their friends.

The LEDP model was introduced by Qin et al. [73] and Imola, Murakami, and Chaudhury [43], with subsequent theoretical expansions [19, 27, 44]. Imola et al. [43, 44] provided the first practical LEDP implementations for triangle and subgraph counting. Recently, Hillebrand et al. [39] improved LEDP triangle counting using hash functions, though their method does not scale to large graphs. All prior LEDP triangle counting algorithms rely on Randomized Response. Imola et al. [43] introduced an LEDP triangle counting algorithm in both non-interactive (single-round) and interactive (multi-round) settings, bounding the standard deviation of the additive error by $O\left(\frac{n^2}{\varepsilon}+\frac{n^{3/2}}{\varepsilon^2}\right)$. In a subsequent work, they reduce the protocol's communication cost [44] by using a combination of sampling and clipping techniques, and refined their standard deviation analysis by using the number of 4-cycles, C_4. Their new theoretical standard deviation is $O\left(\frac{\sqrt{C_4}}{\varepsilon}+\frac{n^{3/2}}{\varepsilon^2}\right)$ for the interactive setting and $O(n^2)$ for the non-interactive setting. Eden et al. [27] further enhanced triangle counting accuracy with an improved post-processing analysis, achieving a standard deviation of $O\left(\frac{\sqrt{C_4}}{\varepsilon^2}+\frac{n^{3/2}}{\varepsilon^3}\right)$ for the non-interactive setting and establishing lower bounds of $\Omega(n^2)$ and $\Omega\left(\frac{n^{3/2}}{\varepsilon}\right)$ for the non-interactive and interactive settings, respectively. Despite these advancements, prior work has neither combined Randomized Response with other privacy mechanisms to improve error bounds nor accounted for input-dependent properties of graphs in theoretical analyses.

For LEDP k-core decomposition, all known algorithms remain theoretical [19]. The algorithm by Dhulipala et al. [19] uses a *level data structure*, where nodes ascend levels based on their noisy induced degrees, with noise drawn from a symmetric geometric distribution to ensure privacy. However, this noise scales with the number of nodes rather than adapting to input structure, leading to significant errors in large graphs. Recent concurrent and independent work by Dhulipala et al. [18] introduces a generalized sparse vector technique to avoid cumulative privacy budget costs, achieving improved theoretical guarantees. However, implementing this approach in a distributed setting is challenging, as it relies on a peeling algorithm that is difficult to distribute. Additionally, the practical performance of these algorithms remains unexplored.

2 PRELIMINARIES

Differential privacy on graphs is defined for *edge-neighboring* inputs. Edge-neighboring inputs are two graphs which differ in exactly one edge. Here, we consider *undirected* graphs.

Definition 2.1 (Edge-Neighboring [69]). *Graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are edge-neighboring if they differ in one edge, namely, if $V_1=V_2$ and the size of the symmetric difference of E_1 and E_2 is 1.* [2]

With high probability (***whp***) is used in this paper to mean with probability at least $1-\frac{1}{n^c}$ for any constant $c \geq 1$.

The local edge differential privacy (LEDP) model assumes that each node in the input graph keeps their adjacency list private. The model is defined in terms of ε-DP algorithms, called *ε-local randomizers (ε-LR)*, that are run individually by every node. The ε-LRs are guaranteed to be ε-DP when the neighboring inputs are adjacency lists that differ in one element. Following [44], we assume that the curator and all nodes act as honest-but-curious adversaries.

Definition 2.2 (ε-Edge Differential Privacy [23, 69]). *Algorithm $\mathcal{A}(G)$, that takes as input a graph G and outputs some value in $Range(\mathcal{A})$,* [3] *is ε-**edge differentially private** (ε-edge DP) if for all $S \subseteq Range(\mathcal{A})$ and all edge-neighboring graphs G and G',*

$$\frac{1}{e^{\varepsilon}} \leq \frac{\Pr[\mathcal{A}(G') \in S]}{\Pr[\mathcal{A}(G) \in S]} \leq e^{\varepsilon}.$$

Definition 2.3 (Local Randomizer (LR) [19]). *An ε-**local randomizer** $R : \mathbf{a} \to \mathcal{Y}$ for node v is an ε-edge DP algorithm that takes as input the set of its neighbors $N(v)$, represented by an adjacency list $\mathbf{a}=(b_1,\ldots,b_{|N(v)|})$. In other words,*

$$\frac{1}{e^{\varepsilon}} \leq \frac{\Pr\left[R(\mathbf{a}') \in Y\right]}{\Pr\left[R(\mathbf{a}) \in Y\right]} \leq e^{\varepsilon}$$

for all $\mathbf{a}$ and $\mathbf{a}'$ where the symmetric difference is 1 and all sets of outputs $Y \subseteq \mathcal{Y}$. The probability is taken over the random choices of R (but not over the choice of the input).

The previous definitions of LEDP [19, 43, 73] are satisfied by the following Definition 2.4. [19] gives a slightly more general and complex definition of LEDP in terms of *transcripts* but all of the algorithms in our paper satisfy our definition below, which is also guaranteed to satisfy their more general transcript-based definition.

Definition 2.4 (Local Edge Differential Privacy (LEDP) [19]). *Given an input graph $G=(V,E)$, for any edge $\{u,v\}$, let algorithm $\mathcal{A}$ assign $\left((R_1^u(\mathbf{a}_u,p_1),\varepsilon_1^u),\ldots,(R_r^u(\mathbf{a}_u,p_r),\varepsilon_r^u)\right)$ to be the set of ε_i^u-local*

[2] The *symmetric difference* of two sets is the set of elements that are in either set, but not in their intersection.

[3] *Range*$(\cdot)$ denotes the set of all possible outputs of a function.

*randomizers called by vertex u during each interactive round and $\left((R_1^v(\mathbf{a}_v, p_1), \varepsilon_1^v), \ldots, (R_s^v(\mathbf{a}_v, p_s), \varepsilon_s^v)\right)$ be the set of ε_i^v-LRs called by v. The private adjacency lists of u and v are given by $\mathbf{a}_u$ and $\mathbf{a}_v$, respectively, and p_i are the new public information released in each round. Algorithm $\mathcal{A}$ is **ε-local edge differentially private (LEDP)** if for every edge, $\{u, v\}$:*

$$\varepsilon_1^u + \cdots + \varepsilon_r^u + \varepsilon_1^v + \cdots + \varepsilon_s^v \leq \varepsilon.$$

For intuition, each LR takes as input the private adjacency list of the node v and public information released in previous rounds; then, it releases new public information for v which will inform the computation of other nodes in the next round. Hence, the algorithm is *interactive*. Each time v releases, it loses some amount of privacy indicated by ε_i^v for the i-th LR. Since edge-neighboring graphs differ in exactly one edge, to ensure the privacy of the system, it is sufficient to ensure that the privacy loss of every edge sums up to ε. Thus, ε-LEDP algorithms also satisfy ε-DP (proven in [19]).

2.1 Privacy Tools

We make use of the following privacy tools and primitives. We define all definitions below in terms of edge-neighboring adjacency lists since our tools will be applied to ε-local randomizers.

Definition 2.5 (Global Sensitivity [23]). *For a function $f : \mathcal{D} \to \mathbb{R}^d$, where $\mathcal{D}$ is the domain of f and d is the dimension of the output, the ℓ_1-**sensitivity** of f is $GS_f = \max_{\mathbf{a},\mathbf{a}'} \|f(\mathbf{a}) - f(\mathbf{a}')\|_1$ for all pairs of $\{\mathbf{a}, \mathbf{a}'\}$ of neighboring adjacency lists (differing in one neighbor).*

Our algorithms and implementations in this paper use the *symmetric geometric distribution* defined in previous papers [6, 13, 23, 24, 29, 81]. The symmetric geometric distribution is also often referred to as the "discrete Laplace distribution." Using this distribution is quite crucial in practice in order to avoid the numerical errors associated with real-valued outputs from continuous distributions.

Definition 2.6 (Symmetric Geometric Distribution [6, 81]). *The **symmetric geometric distribution**, denoted* Geom(b), *with input parameter $b \in (0, 1)$, takes integer values i where the probability mass function at i is given by $\frac{e^b-1}{e^b+1} \cdot e^{-|i| \cdot b}$.*

We denote a random variable drawn from this distribution by $X \sim$ Geom(b). ***With high probability*** (***whp***) is used in this paper to mean with probability at least $1 - \frac{1}{n^c}$ for any constant $c \geq 1$. As with all DP algorithms, privacy is *always* guaranteed and the approximation factors are guaranteed whp. We can upper bound the symmetric geometric noise whp using the following lemma.

Lemma 2.7 ([6, 13, 19, 23, 24, 29, 81]). *With probability at least $1 - \frac{1}{n^c}$ for any constant $c \geq 1$, we can upper bound $X \sim$* Geom (x) *by $|X| \leq \frac{c \ln n}{x}$.*

The ***geometric mechanism*** is defined as follows.

Definition 2.8 (Geometric Mechanism [6, 13, 23, 24]). *Given any function $f : \mathcal{D} \to \mathbb{Z}^d$, where $\mathcal{D}$ is the domain of f and GS_f is the ℓ_1-sensitivity of f, the geometric mechanism is defined as $\mathcal{M}(\mathbf{a}, f(\cdot), \varepsilon) = f(\mathbf{a}) + (Y_1, \ldots, Y_d)$, where $Y_i \sim$* Geom(ε/GS_f) *are independent and identically distributed (i.i.d.) random variables drawn from* Geom(ε/GS_f) *and* $\mathbf{a}$ *is a private input adjacency list.*

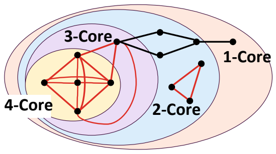


Figure 2: Example k-core decomposition and triangles in a 4-degenerate graph. Nodes are assigned core numbers based on the highest value core they belong to; e.g., a node in the 1-core but not in the 2-core is given the core number of 1. Larger valued cores are contained within all smaller valued cores; e.g., the 3-core is contained in the 1 and 2-core. Red edges show the triangles, i.e., 3-cycles in the graph. The degeneracy of this graph is 4.

Definition 2.9 (Laplace Distribution). *The probability density function of the Laplace distribution on $X \in \mathbb{R}$ is $Lap(b) = 2b \cdot \exp\left(-(|X| \cdot b)\right)$*

Lemma 2.10 (Laplace Mechanism [23]). *Given a function $f : \mathcal{G} \to R$ with sensitivity Δ_f, $f(\mathcal{G}) + Lap\left(\frac{\varepsilon}{\Delta_f}\right)$ is ε-differentially private.*

Lemma 2.11 (Privacy of the Geometric Mechanism [6, 13, 23, 24]). *The geometric mechanism is ε-DP.*

In addition to the Geometric Mechanism, our paper also uses Randomized Response (RR). Randomized Response (RR) when applied to graphs flips the bit indicating the existence of an edge in the graph. We define RR in terms of the way it is used on graphs.

Definition 2.12 (Randomized Response). *Randomized response on input graph $G = (V, E)$, represented as an upper triangular adjacency matrix M which only contains entries $M[i, j]$ where $i < j$, flips every bit $M[i, j]$ (where $i < j$) in the matrix with probability $\frac{1}{e^\varepsilon+1}$.*

It is well-known that randomized response is ε-DP.

Lemma 2.13 ([23]). *Randomized response is ε-DP.*

The *composition theorem* guarantees privacy for the *composition* of multiple algorithms with privacy guarantees of their own. In particular, this theorem covers the use case where multiple LEDP algorithms are used on the *same* dataset.

Theorem 2.14 (Composition Theorem [22, 23, 25]). *A sequence of DP algorithms, $(\mathcal{A}_1, \ldots, \mathcal{A}_k)$, with privacy parameters $(\varepsilon_1, \ldots, \varepsilon_k)$ form at worst an $(\varepsilon_1 + \cdots + \varepsilon_k)$-DP algorithm under* adaptive composition *(where the adversary can adaptively select algorithms after seeing the output of previous algorithms).*

Finally, the post-processing theorem states that the result of post-processing on the output of an ε-LEDP algorithm is ε-LEDP.

Theorem 2.15 (Post-Processing [12, 23]). *Let $\mathcal{M}$ be an ε-LEDP algorithm and h be an arbitrary (randomized) mapping from $Range(\mathcal{M})$ to an arbitrary set. The algorithm $h \circ \mathcal{M}$ is ε-LEDP.*[4]

We use implementations by the Google privacy team [87], which also guarantee cryptographic security.

2.2 Problem Definitions

Below, we define the k-core decomposition, low out-degree ordering, and triangle counting problems that we study.

In this paper, we consider undirected graphs $G = (V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. We use $[n]$ to denote $\{1, \ldots, n\}$. For ease of indexing, we set the IDs of V to be $V = [n]$. The set of neighbors of a node $i \in [n]$ is denoted by $N(i)$, and the degree of node i is denoted $\deg(i)$.

[4] $\circ$ is notation for applying h on the outputs of $\mathcal{M}$.

Definition 2.16 (k-Core Decomposition). *Given an input graph, $G = (V, E)$, a k-**core** is a maximal induced subgraph in G where every node has degree at least k. A k-**core decomposition** assigns a **core number** to each node $v \in V$ equal to κ if v is in the κ-core but not the $(\kappa+1)$-core. Let $k(v)$ be the core number of v.*

See Fig. 2 for an example. No *exact* k-core decomposition algorithm satisfies the definition of DP (or LEDP). Hence, our algorithms take an input graph G and output an *approximate* core number for each node in the graph (Definition 2.17) and an approximate ***low out-degree ordering*** (Definition 2.19).

Definition 2.17 ((ϕ, ζ)-Approximate Core Number [19]). *Let $\hat{k}(v)$ be an approximation of the core number of v, and let $\phi \geq 1, \zeta \geq 0$. The core estimate $\hat{k}(v)$ is a (ϕ, ζ)-**approximate core number** of v if $k(v) - \zeta \leq \hat{k}(v) \leq \phi \cdot k(v) + \zeta$.*

We define the related concept of an ***approximate low out-degree ordering*** based on the definition of ***degeneracy***.

Definition 2.18 (Degeneracy). *An undirected graph $G = (V, E)$ is d-degenerate if every induced subgraph of G has a node with degree at most d. The* degeneracy *of G is the smallest value of d for which G is d-degenerate.*

It is well known that degeneracy $d = \max_{v \in V}\{k(v)\}$.

Definition 2.19 ((ϕ, ζ)-Approximate Low Out-Degree Ordering). *Let $D = [v_1, v_2, \ldots, v_n]$ be a total ordering of nodes in a graph $G = (V, E)$. The ordering D is an (ϕ, ζ)-**approximate low out-degree ordering** if orienting edges from earlier nodes to later nodes in D produces out-degree at most $\phi \cdot d + \zeta$.*

Definition 2.20 (Triangle Count). *Given an undirected input graph $G = (V, E)$, the triangle count returns the number of 3-cycles in G.*

2.3 Distributed Simulation Model

Local Edge Differential Privacy (LEDP) is inherently decentralized: each user (or node) independently perturbs their private local data (adjacency list) before any communication. This model aligns naturally with distributed systems, where data is often siloed across machines or clients. To evaluate LEDP algorithms in such settings, we adopt a distributed simulation framework that closely mirrors real-world deployments. Specifically, we use a coordinator-worker model in which each worker is assigned a partition of nodes along with their full adjacency lists. Workers execute LEDP algorithms locally and communicate only privacy-preserving outputs to a central coordinator. The coordinator aggregates these responses and broadcasts public updates to all workers, proceeding iteratively over synchronous communication rounds. While assigning one machine per node is infeasible at scale, this simulation preserves the privacy and communication structure of LEDP and allows for practical evaluation of network overhead on large graphs.

3 PRACTICAL k-CORE DECOMPOSITION ALGORITHM

We present k-CoreD, a novel k-core decomposition algorithm that addresses key limitations of prior work [19] through principled algorithmic design. While their framework offers strong theoretical foundations under the ϵ-LEDP model, its dependence on the total number of nodes leads to large additive error and excessive communication rounds. Our algorithm replaces this dependency with input-sensitive parameters—specifically, the graph's maximum degree—through *degree thresholding* and *bias terms*. These techniques yield improved asymptotic bounds and significantly lower empirical error, as confirmed by our experiments.

3.1 Algorithm Description

Our algorithm operates synchronously over $O(\log(n)\log(D_{\max}))$ rounds, where $D_{\max}$ is the maximum degree of the graph. The algorithm outputs $\left(2+\eta, O\left(\frac{\log(D_{\max})\log^2(n)}{\epsilon}\right)\right)$-approximate core numbers with high probability, as well as a low out-degree ordering with the same approximation guarantee. Throughout this section, the term $\log(n)$ denotes $\log_{1+\psi}(n)$, unless explicitly stated otherwise (where ψ is a constant). The algorithm uses a level data structure (LDS) [19], where nodes are assigned levels that are updated iteratively. Levels are partitioned into groups of equal size, with each group g_i containing $\frac{\lceil\log(n)\rceil}{4}$ consecutive levels. We limit the number of rounds a node participates in, based on its noisy degree, which we refer to as degree thresholding. This significantly reduces the number of rounds, from $O(\log^2(n))$ [19] to $O(\log(n)\log(D_{\max}))$. In each round, the algorithm uses a noisy count of a node's neighbors at the same level to decide if it should move up a level. After processing all nodes in a round, the updated LDS is published for use in subsequent rounds. Once all rounds are complete, the algorithm estimates the core numbers of the nodes based on their final levels, using Algorithm 3.4. Additionally, a low out-degree ordering is determined by sorting nodes from smaller to larger levels, breaking ties using node IDs.[5]. The algorithm is implemented in a distributed setting, where computation is divided between a coordinator and multiple workers. The pseudocode is structured to reflect this division. We now describe their respective roles.

Coordinator. As described in Algorithm 3.1, the coordinator takes as input the graph size n, number of workers M, constant $\psi > 0$, privacy parameter $\varepsilon \in (0, 1]$, privacy split fraction $f \in (0, 1)$, and bias term b. It first computes the privacy budgets ε_1 and ε_2 for degree thresholding and noisy neighbor counts, respectively, based on ε and f (Line 5). The coordinator maintains the *level data structure* (LDS), where $LDS[i]$ stores the current level of node i, and a communication channel, *channel*, for receiving messages from the workers. All nodes are initialized to level 0 (Line 7) and are incrementally moved up in later rounds based on signals received from the workers. It begins by signaling the workers to load their assigned subgraphs in parallel (Line 10) and then collects the degree threshold values to determine the total number of rounds (Line 11). In each round r, it computes the corresponding group index (Line 13) and launches M asynchronous worker processes (Line 15). Each worker returns a bit vector indicating whether each node in its subgraph should move up a level. After all processes complete (Line 16), the coordinator processes the responses and updates the LDS accordingly. It then publishes the new LDS (Line 21) before the next round begins. After the final round, the coordinator computes the estimated core numbers using the final LDS.

Worker (Degree Thresholding). As shown in Algorithm 3.2, each worker begins by loading its assigned subgraph and initializing local structures. For each node v, it computes a *noisy degree* $\widetilde{d}_v = d_v + X$,

[5] This doesn't leak privacy, as IDs are assigned to nodes and not edges and reveal no information about the sensitive edge data.

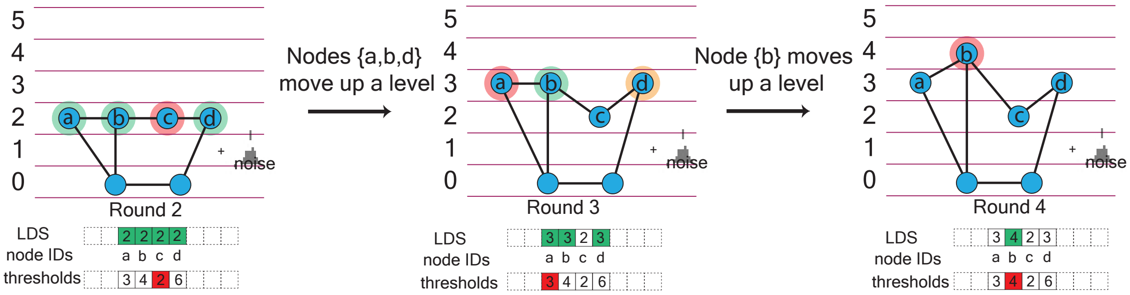


Figure 3: Node movements in k-CoreD's Level Data Structure (LDS). Green: active nodes eligible to move; red: thresholded nodes; orange: active nodes that fail the noisy neighbor check. The LDS and threshold structures are shown alongside the graph. Noise is added during the level-moving step to ensure privacy, and snapshots illustrate node progression and halted movement due to thresholding.

where d_v is the true degree and $X \sim \text{Geom}(\frac{\varepsilon}{2})$ is symmetric geometric noise (Line 6). To mitigate large positive noise and reduce overestimation, a bias term—proportional to the noise's standard deviation—is subtracted from $\widetilde{d}_v$. The worker then computes a threshold value for each node as $\left\lceil \log_2(\widetilde{d}_v) \right\rceil \cdot L$, where L is the number of levels per group in the LDS (Line 8). These thresholds determine the maximum number of rounds in which each node participates. The worker keeps track of the maximum threshold across its subgraph and returns it to the coordinator (Line 11).

Worker (Level Moving). In each round, workers assess whether nodes in their subgraph should move up a level. As shown in Algorithm 3.3, if a node v has already reached its threshold round r, it is skipped (Line 5). Otherwise, if v is currently at level r, we count the number of its neighbors that are also at the same level (Line 9). To ensure privacy, this count $\mathcal{U}_v$ is perturbed to produce a noisy estimate $\widehat{\mathcal{U}}_v = \mathcal{U}_v + X + B$ (Line 13), where X is symmetric geometric noise with parameter $s = \frac{\varepsilon}{2 \cdot v.threshold}$ and B is an added bias term to counteract large negative noise. The node moves up a level if $\widehat{\mathcal{U}}_v > (1+\eta/5)^{\mathcal{F}(r)}$, where $\mathcal{F}(r)$ is the group index corresponding to the current level (Line 15). After processing all nodes, the worker sends the updated level-change bits to the coordinator (Line 18).

Bias Terms. We introduce two analytically derived bias terms, based on the standard deviation of the symmetric geometric distribution—one for degree thresholding and one for level movement. The first bias term is subtracted from the computed threshold to account for situations where a large positive noise is chosen. If a large positive noise is chosen, we lose privacy proportion to the new threshold in the level moving step. Hence, our bias term biases the result to smaller thresholds, resulting in less privacy loss.

The second bias term is added to the computed induced noisy degree to account for situations where a large negative noise prevents nodes from moving up the first few levels of the structure. Our added bias allows nodes with non-zero degrees to move up the first levels of the structure. Since the degree bounds increase exponentially, this additional bias term accounts for smaller errors as nodes move up levels. Notably, we observe this behavior in our experiments when comparing to the baseline implementation of [19], which omits the bias term: many nodes remain stuck at level 0, resulting in significantly higher error. This highlights the practical importance of our bias correction.

Example 3.1. *Fig. 3 illustrates node movement in the LDS during k-CoreD. In each round, nodes compute a noisy count of same-level neighbors and move up if it exceeds a threshold based on group index—unless blocked by their degree threshold. In the LDS, green marks eligible nodes; red in the threshold array marks those blocked; and in the graph view, green means movement and red/orange means restriction. For instance, node c is blocked in Round 1, a in Round 2, and b in Round 3. In Round 2, node d is not thresholded but remains at the same level due to failing the noisy neighbor check (Algorithm 3.3 Line 14).*

Algorithm 3.1: k-Core Decomposition and Ordering (Coordinator)

```
Input: graph size $n$; number of workers $M$; approx constant
  $\psi \in (0,1)$; privacy parameter $\varepsilon \in (0,1]$; split fraction $f \in (0,1)$;
  bias term $b$.
Output: Approximate core numbers and low out-degree ordering
  of each node in $G$.
Function $k$-CoreD($n, \psi, \varepsilon, f, b$)
    Set $\lambda = \frac{(5-2\eta)\eta}{(\eta+5)^2}$; $L = \frac{\lceil \log n \rceil}{4}$
    Set $\varepsilon_1 = f \cdot \varepsilon$ and $\varepsilon_2 = (1-f) \cdot \varepsilon$
    Set $C \leftarrow$ new Coordinator ($LDS, channel$)
    Coordinator initializes $C.LDS$ with $C.LDS[i] \leftarrow 0 \; \forall i \in [n]$.
    Set $maxDegreeThresholds \leftarrow [\,]$
    parfor $w = 1$ to $M$ do
        $maxDegreeThresholds[w]$ =
          DegreeThresholdWorker($w, \varepsilon_1, L, b$)
    Set $numOfRounds$=
      $\min\left(4\log(n)\log(\widetilde{d}_{max}) - 1, \max\left(maxDegreeThresholds\right)\right)$
    for $r = 0$ to $numOfRounds$ do
        Set $\mathcal{F}(r) \leftarrow \lfloor \frac{r}{L} \rfloor$
        parfor $w = 1$ to $M$ do
            LevelMovingWorker ($w, r, \varepsilon_2, \psi, \mathcal{F}(r), C.LDS$)
        $C$.wait()        ▹ coordinator waits for workers to finish
        $nextLevels \leftarrow C.channel$
        for $i = 1$ to $n$ do
            if $nextLevels[i] = 1$ then
                $C.LDS$.levelIncrease ($i$)
        Coordinator publishes updated $C.LDS$
    Coordinator calls
      $cores \leftarrow C$.EstimateCoreNumbers($C.LDS, L, \lambda, \psi$)
    Coordinator produces $D$, a total order of all nodes, using levels
      from $C.LDS$ (from smaller to larger) breaking ties by node ID
    Return ($cores, D$)
```

Algorithm 3.2: Degree Thresholding (Worker)

```
Input: worker ID $w$; privacy parameter $\varepsilon \in (0, 1]$; levels per group $L$; bias term $b$.
Function DegreeThresholdWorker($w, \varepsilon, L, b$)
    Set $maxThreshold \leftarrow 0$
    for $node\ \ v := localGraph$ do
        Sample $X \sim \mathrm{Geom}\left(\frac{\varepsilon}{2}\right)$
        $\widetilde{d_v} \leftarrow d_v + X$                                  ▷ noised degree
        $\widetilde{d_v} \leftarrow \widetilde{d_v} + 1 - \min\left(b \cdot \frac{2 \cdot e^{\varepsilon}}{e^{2\varepsilon} - 1}, \widetilde{d_v}\right)$
        $v.threshold \leftarrow \left\lceil \log_2(\widetilde{d_v}) \right\rceil \cdot L$                ▷ thresholding
        $v.permZero \leftarrow 1$
        $maxThreshold = \max(maxThreshold, v.threshold)$
    $w$.send ($maxThreshold$)
```

Algorithm 3.3: Level Moving (Worker)

```
1  Input: worker id $w$; round number $r$; privacy parameter $\varepsilon \in (0, 1]$;
   constant $\psi$; group index $\mathcal{F}(r)$; pointer to the coordinator $LDS$.
2  Function LevelMovingWorker($w, r, \varepsilon, \psi, \mathcal{F}(r), LDS$)
3      Set $nextLevels \leftarrow [0, \ldots, 0]$
4      for $node\ \ v := localGraph$ do
5          if $v.threshold = r$ then
6              $v.permZero = 0$
7          $vLevel \leftarrow LDS$.getLevel ($v$)
8          if $vLevel = r$ and $v.permZero \neq 0$ then
9              Let $\mathcal{U}_v$ be the number of neighbors $j \in \mathbf{a}_v$ where
               $LDS$.getLevel ($j$) $= r$.
10             Set scale $s \leftarrow \frac{\varepsilon}{2 \cdot (v.\text{threshold})}$
11             Sample $X \sim \mathrm{Geom}(s)$.
12             Set extra bias $B \leftarrow \frac{6e^s}{(e^{2s}-1)^3}$
13             Compute $\widehat{\mathcal{U}_v} \leftarrow \mathcal{U}_v + X + B$.
14             if $\widehat{\mathcal{U}_i} > (1 + \eta/5)^{\mathcal{F}(r)}$ then
15                 $nextLevels[v] = 1$
16             else
17                 $v.permZero = 0$
18     $w$.send ($w$, $nextLevels$)
19     $w$.done ()
```

Algorithm 3.4: Estimate Core Number (Coordinator) [55]

```
Function EstimateCoreNumbers($LDS, L, \lambda, \eta$)
    for $i = 1$ to $n$ do
        $\hat{k}(i) \leftarrow (2 + \lambda)(1 + \eta/5)^{\max\left(\left\lfloor \frac{LDS[i]+1}{L} \right\rfloor - 1, 0\right)}$.
    Return $\{(i, \hat{k}(i)) : i \in [n]\}$
```

3.2 Theoretical Analysis

Memory Analysis & Communication Cost. Let M be the number of workers and n the graph size. Each worker processes S nodes, where $S = \lfloor n/M \rfloor$ for $M - 1$ workers, and the last worker handles $n-(M-1)\lfloor n/M \rfloor$. The coordinator maintains the level data structure (LDS) and a communication channel, both requiring $O(n)$ space, resulting in a total memory usage of $O(n)$. Each worker processes $O(S)$ nodes, requiring $O(Sn)$ space for the graph and an additional $O(S)$ space for auxiliary structures, leading to a total of $O(Sn)$. In terms of communication, workers send one bit per node per round, incurring a per-worker cost of $O(S)$ and an overall round cost of $O(n)$. The coordinator receives and distributes the updated LDS, adding another $O(n)$ cost. Thus, the total communication overhead for the algorithm is $O(n \log(n) \log(D_{\max}))$.

Privacy Guarantees. Our privacy guarantees depend on the following procedures. First, we perform degree-based thresholding, which upper bounds the number of levels a node can move up. Second, we subtract and add bias terms to the results of our mechanisms. And finally, we scale our noise added in Line 10 of Algorithm 3.3 by the noisy threshold. We show that our algorithm can be implemented using local randomizers (Definition 2.3). Then, we show that the local randomizers have appropriate privacy parameters to satisfy ε-LEDP (Definition 2.4).

Lemma 3.2 (Degree Threshold LR)**.** *Our degree thresholding procedure run with privacy parameter ε' is a $(\varepsilon'/2)$-local randomizer.*

Proof. Our degree-thresholding procedure upper bounds the number of levels that we iterate through using the (private) degree of each node. Specifically, it adds symmetric geometric noise to the degree $\widetilde{d_u} = d_u + \mathrm{Geom}(\varepsilon'/2)$ and then computes $\lceil \log_{1+\eta}(\widetilde{d_u}) \rceil \cdot L$, where L is the number of levels per group. The sensitivity of the degree of any node is 1 and by the privacy of the geometric mechanism (Lemma 2.11), the output $\widetilde{d_u}$ is $(\varepsilon'/2)$-DP. Then, producing the final level upper bound uses post-processing (Theorem 2.15) where privacy is preserved. Hence, our output is $(\varepsilon'/2)$-DP and the algorithm can be implemented as a $(\varepsilon'/2)$-local randomizer. □

Using Lemma 3.2, we prove Theorem 3.3.

Theorem 3.3. *Algorithm 3.1 is ε-LEDP.*

Proof. Our algorithm calls the local randomizers in Lemma 3.2 with $\varepsilon_1 = \varepsilon \cdot f$, where $f \in (0, 1)$ is a fraction which splits some portion of the privacy budget, and then iterates through the levels one-by-one while adding noise to the induced degree of each node consisting of all neighbors of the node on the same or higher level. We showed in Lemma 3.2 that the degree thresholding procedure can be implemented as $(\varepsilon_1/2)$-local randomizers.

The key to our better error bounds is our upper bound on the number of levels we iterate through, bounded by our threshold. Since the thresholds are public outputs from the local randomizers, we can condition on these outputs. Let the threshold picked for node v be denoted as t_v. Then, we add symmetric geometric noise to the induced degree of the node (among the neighbors at or above v's current level) drawn from Geom $(\varepsilon_2/(2 \cdot t_v))$ where $\varepsilon_2 = \varepsilon \cdot (1 - f)$. Conditioning on the public levels of each node, the sensitivity of the induced degree of any node is 1. By the privacy of the geometric mechanism, we obtain a $(\varepsilon_2/(2 \cdot t_v))$-local randomizer for v. By composition (Theorem 2.14) over at most t_v levels, the set of all local randomizers called on v, is $(\varepsilon_2/2)$-differentially private. For any edge, the sum of the privacy parameters of the set of all local randomizers called on the endpoints of the edge is $2 \cdot \varepsilon_1/2 + 2 \cdot \varepsilon_2/2 = \varepsilon_1 + \varepsilon_2 = f \cdot \varepsilon + (1 - f) \cdot \varepsilon$. By Definition 2.4, this is ε-LEDP.

Finally, our bias terms, added or subtracted after applying the geometric mechanism, preserve privacy due to the post-processing invariance of differential privacy (Theorem 2.15). □

Approximation Guarantees. Our algorithm given in the previous section contains several changes that results in better theoretical bounds and optimizes the practical performance on real-world datasets. To prove our approximation factors, we first show Invariant 1 and Invariant 2 hold for our modified algorithm. $D_{\max}$ is the graph's max degree.

Invariant 1 (Degree Upper Bound [19]). *If node $i \in V_r$ (where V_r contains nodes in level r) and $r < 4\log^2 n - 1$, then i has at most $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor} + \frac{c\log(D_{\max})\log^2(n)}{\varepsilon}$ neighbors in levels $\geq r$, with high probability, for constant $c > 0$.*

Invariant 2 (Degree Lower Bound [19]). *If node $i \in V_r$ (where V_r contains nodes in level r) and $r > 0$, then i has at least $(1 + \eta/5)^{\lfloor (r-1)/(2\log n)\rfloor} - \frac{c\log(D_{\max})\log^2(n)}{\varepsilon}$ neighbors in levels $\geq r - 1$, with high probability, for constant $c > 0$.*

There are two parts to our analysis: first, we prove that our degree thresholding procedure does not keep the node at too low of a level, with high probability; second, we show that our new procedure for moving up levels adds at most the noise used in [19] and not more. Together, these two arguments maintain the invariants.

Lemma 3.4. *Degree-thresholding satisfies Invariant 1.*

Proof. By Lemma 2.7, the noise we obtain for thresholding is upper bounded by $\frac{c'\ln n}{(\varepsilon_1/2)} = \frac{2c'\ln n}{f\cdot\varepsilon}$ with probability at least $1 - \frac{1}{n^{c'}}$. Thus, the noisy degree $\tilde{d_v}$ we obtain in Line 6 of Algorithm 3.2 follows $\tilde{d_v} \geq d_v - \frac{2c'\ln n}{f\cdot\varepsilon}$ with probability at least $1 - \frac{1}{n^{c'}}$. Let r be the level we output as the threshold level. Then, we can compute the upper degree bound of this level to be at least $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor} = (1+\eta/5)^{\log_{1+\eta/5}(\tilde{d_v})} \geq (1+\eta/5)^{\log_{1+\eta/5}\left(d_v - \left(\frac{2c'\ln n}{f\varepsilon}\right)\right)} = d_v - \frac{2c'\ln n}{f\varepsilon}$. The maximum induced degree of v on any level is at most d_v. Let $d_{v,r}$ be the induced degree of v on the thresholded level r, it must hold that $d_{v,r} \leq \left(d_v - \frac{2c'\ln n}{f\varepsilon}\right) + \frac{4c'\ln n}{f\varepsilon} + \frac{2c_3\ln n}{f\varepsilon} \leq (1+\eta/5)^{\lfloor r/(2\log n)\rfloor} + \frac{(4c'+2c_3)\ln n}{f\varepsilon}$. Since f and c' are both constants and Invariant 1 allows for picking a large enough constant $c > 0$, we can pick $c \geq 2(c' + c_3)/f$ and Invariant 1 is satisfied where c' and c_3 are fixed constants ≥ 1. □

We do not have to prove that our thresholded level satisfies Invariant 2 since we use the threshold level as an *upper bound* of the maximum level that a node can be on. Hence, a node will not reach that level unless the procedure for moving the node up the levels satisfies Invariant 2. We now prove that our level movement procedure satisfies both invariants.

Lemma 3.5. *Our level moving algorithm satisfies Invariant 1 and Invariant 2.*

Proof. Our level moving algorithm is similar to [19] except that we pick noise based on the threshold. Thus, by Lemma 2.7, our algorithm picks noise that is at most $\frac{2c_1\cdot t_v\ln n}{\varepsilon}$ with probability at least $1 - \frac{1}{n^{c_1}}$ where t_v is the released threshold for v and $c_1 \geq 1$ is a fixed constant. Also, by Lemma 2.7, it holds that $t_v \leq \log\left(D_{\max} + \frac{2c_2\ln n}{f\varepsilon} - \frac{2c_3\ln n}{f\varepsilon}\right)\cdot L \leq 2\log(D_{\max})\log n$ with probability at least $1 - \frac{1}{n^{c_2}}$. A node moves up a level from level r if its induced degree plus the noise exceeds the threshold $(1 + \eta/5)^{\lfloor r/(2\log n)\rfloor}$. Thus, using our computed noise, the degree of a node must be at least $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor} - 2\log(D_{\max})\left(\frac{2c_1\log^2 n}{\varepsilon}\right)$ with probability at least $1 - \frac{1}{n^{c_1}} - \frac{1}{n^{c_2}}$ when it moves up from level r. By choosing large enough constants $c_1, c_2 > 0$ and $c \geq 4c_1c_2$, we satisfy Invariant 2. Similarly, the node does not move up from level r when its induced degree plus noise is at most $(1+\eta/5)^{\lfloor r/(2\log n)\rfloor}$. By a symmetric argument to the above, we show that Invariant 1 is satisfied. □

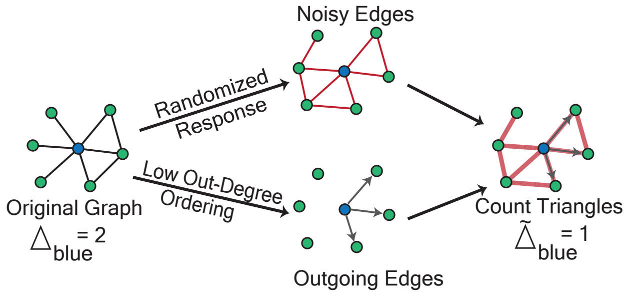


Figure 4: EdgeOrient$_\triangle$ for blue node: true triangle count is 2, but due to Randomized Response and low out-degree ordering, estimate is 1.

Finally, using Lemma 3.4 and Lemma 3.5, we can prove the final approximation factor for our algorithm.

Theorem 3.6. *Our algorithm returns $(2+\eta, O(\log(D_{\max})\log^2(n)/\varepsilon))$-approximate k-core numbers, with high probability, in $O(\log(n)\log(D_{\max}))$ rounds of communication.*

Proof. By Lemma 3.4 and Lemma 3.5, our algorithm satisfies Invariant 1 and Invariant 2, hence, our algorithm returns our desired approximation using Theorem 4.1 of [19]. Our algorithm iterates to value at most $O(\log n\log(D_{\max}))$ for $D_{\max}$ (the maximum threshold), resulting in a total of $O(\log n\log(D_{\max}))$ rounds. □

4 TRIANGLE COUNTING USING LOW OUT-DEGREE ORDERING

We present our novel triangle counting algorithm, EdgeOrient$_\triangle$, which leverages the low out-degree ordering obtained from the k-CoreD algorithm along with randomized response (RR). Prior works rely on either all or a sample of neighboring edges after applying RR, and often suffer from error bounds that scale poorly with graph size. To address these limitations, our algorithm exploits a new input-dependent graph property, the degeneracy (maximum core number), to upper bound the number of oriented 4-cycles (Fig. 5), yielding significantly tighter error bounds in theory and practice.

4.1 Algorithm Description

Our algorithm consists of three additional computation rounds after computing the low out-degree ordering (Z) using k-CoreD. In the first round, each node perturbs its adjacency list using Randomized Response (RR) [94], producing a privacy-preserving set of noisy edges for subsequent computations. In the second round, we calculate the maximum noisy out-degree, $\widetilde{d}_{\max}$, by determining each node's outgoing edges based on Z. While the first two rounds can be combined, we separate them for clarity. In the final round, we compute the number of triangles incident to each node, using the noisy edges and the maximum noisy out-degree, $\widetilde{d}_{\max}$. The algorithm is

Algorithm 4.1: ε-LEDP Triangle Counting (Coordinator)

```
1  Input: graph size $n$; number of workers $M$; constant $\psi \in (0,1)$;
     privacy parameter $\varepsilon \in (0,1]$; split fraction $f \in (0,1)$; bias term $b$;
2  Output: Noisy Triangle Count
3  Function EdgeOrient$_\Delta(n, M, \psi, \varepsilon, f, b)$
4      Set $Z \leftarrow k$-CoreD$(n, \psi, \frac{\varepsilon}{4}, f, b)$ (Algorithm 3.1)
5      Set $C \leftarrow$ new Coordinator $(cRR, cTCount, cMaxOut, X)$
6                                             ▷ Round 1: Randomized Response
7      parfor $w = 1$ to $M$ do
8          RRWorker $\left(w, n, \frac{\varepsilon}{4}\right)$
9      $C$.wait()                 ▷ coordinator waits for workers to finish
10     $C$.publishNoisyEdges$(cRR, X)$
11                                            ▷ Round 2: Max Out-degree
12     parfor $w = 1$ to $M$ do
13         MaxOutDegreeWorker $\left(w, \frac{\varepsilon}{4}, Z\right)$
14     $C$.wait()
15     $\widetilde{d}_{\max} \leftarrow \max\left(\{C.cMaxOut[i] \mid i \in [M]\}\right) + \frac{12\log(n)}{\varepsilon}$
16                                            ▷ Round 3: Count Triangles
17     parfor $w = 1$ to $M$ do
18         CountTrianglesWorker $\left(w, \frac{\varepsilon}{4}, Z, X, \widetilde{d}_{\max}\right)$
19     $C$.wait()
20     $\widetilde{\Delta} \leftarrow \sum_{i=1}^{n} C.cTCount\,[i]$
21     Return $\widetilde{\Delta}$
```

Algorithm 4.2: Randomized Response (Worker)

```
1  Input: worker id $w$; graph size $n$; privacy parameter $\varepsilon \in (0,1]$;
2  Function RRWorker$(w, n, \varepsilon)$
3      Set $neighborsRR \leftarrow [\,][\,]$
4      for node $v := localGraph$ do
5          For all $ngh_v \leftarrow \{j : j \in [n] \wedge j > v\}$
6          $neighborsRR[v] = \text{RandomizedResponse}_\varepsilon(ngh_v)$
7      $w$.sendRR$(w, neighborsRR)$
8      $w$.done ()
```

Algorithm 4.3: Noisy Max Out-Degree (Worker)

```
1  Input: worker id $w$; graph size $n$; privacy parameter $\varepsilon \in (0,1]$;
2  Function MaxOutDegreeWorker$(w, \varepsilon, Z)$
3      Set $out_{\max} \leftarrow 0.$
4      for node, adjacency list $v, ngh := localGraph$ do
5          Set $ngh_v \leftarrow \{j : j \in ngh \wedge Z[j] > Z[v]\}$
6          $B \sim \text{Geom}(\varepsilon)$
7          $out_{\max} \leftarrow \max(out_{\max}, |ngh_v| + B)$
8      $w$.sendMaxOutdegree$(out_{\max})$
9      $w$.done ()
```

Algorithm 4.4: Triangle Counting (Worker)

```
1  Input: worker id $w$; privacy parameter $\varepsilon \in (0,1]$; low out-degree
     ordering $Z$, published noisy edges $X$; public maximum noisy
     out-degree $\widetilde{d}_{\max}$;
2  Function CountTrianglesWorker$(w, \varepsilon, Z, X, \widetilde{d}_{\max})$
3      Set $workerTCount \leftarrow 0.0$
4      for node, adjacency list $v, ngh := localGraph$ do
5          Set $\widetilde{\Delta} \leftarrow 0.0$
6          $\text{OutEdges}_v = \{j : j \in ngh \wedge Z\,[j] > Z\,[v]\}$
7          for $i_1 \in \{1, \dots, \min(\widetilde{d}_{\max}, |OutEdges_v|)\}$ do
8              for $i_2 \in \{i_1 + 1, \dots, \min(\widetilde{d}_{\max}, |OutEdges_v|)\}$ do
9                  $j \leftarrow \text{OutEdges}_v[i_1]$
10                 $k \leftarrow \text{OutEdges}_v[i_2]$
11                 $\widetilde{\Delta} \leftarrow \widetilde{\Delta} + \frac{X_{\{j,k\}} \cdot (e^{\varepsilon}+1) - 1}{e^{\varepsilon} - 1}$
12         Sample $R \sim \text{Lap}\left(\frac{\varepsilon}{2 \cdot \widetilde{d}_{\max}}\right)$
13         $\widetilde{\Delta} \leftarrow \widetilde{\Delta} + R$
14         $workerTCount \leftarrow workerTCount + \widetilde{\Delta}$
15     $w$.sendTCount $(w, workerTCount)$
16     $w$.done ()
```

implemented in a distributed setting, where computation is divided between a coordinator and multiple workers. The pseudocode is structured to reflect this division.

Coordinator As shown in Algorithm 4.1, the coordinator receives the graph size n, number of workers M, constant parameter $\psi > 0$, privacy parameter $\varepsilon \in (0,1]$, privacy split fraction $f \in (0,1)$, and bias term b. It initializes three channels, cRR, $cMaxOut$, $cTCount$, to receive RR noisy edges, maximum noisy out-degrees, and local noisy triangle counts from workers (Line 5). The coordinator manages the algorithm's execution, collecting worker outputs and publishing updates each round. It first computes the low out-degree ordering, Z, using k-CoreD. In the first round, it launches M asynchronous worker processes (Line 6), each computing and sending noisy edges after RR. Before the second round, the coordinator aggregates and stores them in X, then publishes X for global access (Line 10), enabling workers to utilize the noisy public edges in subsequent computations. In the second round (Line 11), workers compute noisy maximum out-degrees for their subgraphs using Algorithm 4.3. The coordinator then determines $\widetilde{d}_{\max}$, the maximum noisy out-degree across all workers (Line 15). Finally, in the third round (Line 16), each worker counts the triangles incident to its nodes using the low out-degree ordering, published noisy edges, and $\widetilde{d}_{\max}$. Workers send noisy local triangle counts to the coordinator, which aggregates them to compute the overall noisy triangle count (Line 20).

Worker (Randomized Response) As specified in Algorithm 4.2, workers maintain a *neighborsRR* data structure to store noisy edges. For each node v, noisy edges are computed via Randomized Response (RR) with parameter ε, processing only the upper triangular part of the adjacency matrix (Line 5), as the graph is undirected. Specifically, for a node v, all indices greater than v are processed using $\text{RandomizedResponse}_\varepsilon(ngh_v)$, which flips the existence of each edge (v, ngh_v) with probability $\frac{1}{e^{\varepsilon}+1}$ (Line 6). Once computed, workers send noisy edges to the coordinator (Line 7).

Worker (Noisy Max Out-Degree) In Algorithm 4.3, workers maintain a variable out_{max} which stores the maximum noisy out-degree of their subgraph. For each node v, the worker first computes the out-degree d_v using the order provided in Z, where an edge (v, j) is considered outgoing if $Z[j] > Z[v]$ (Line 5). The out-degree d_v is the number of outgoing edges from v. Then, the worker adds symmetric geometric noise with parameter ε to d_v, computes the max noisy out-degree, and sends $out_{\max}$ to the coordinator (Line 8).

Worker (Count Triangles) As shown in Algorithm 4.4, each worker computes the number of triangles incident to each node in its respective subgraph. For each node v, the outgoing edges are identified and sorted in ascending order by node IDs. The triangle count is determined by iterating over all unique pairs of outgoing neighbors $\{j, k\}$ of v, up to $\widetilde{d}_{\max}$ (Line 7,8). For each pair, the triangle contribution is calculated as: $\frac{X_{\{j,k\}} \cdot (e^{\varepsilon}+1)-1}{e^{\varepsilon}-1}$, where $X_{\{j,k\}}$ represents the noisy presence (1) or absence (0) of an edge between j and k (Line 11). To ensure privacy, additional noise is added to the triangle counts using the Laplace distribution[6] with parameter $\frac{\varepsilon}{2 \cdot \widetilde{d}_{\max}}$, where $\widetilde{d}_{\max}$ is the global maximum noisy out-degree. Upon completing the computation, each worker aggregates and returns the noisy triangle count for its entire subgraph (Line 15).

Example 4.1. *In Fig. 4, we apply EdgeOrient$_\Delta$ to estimate the number of triangles incident to the blue node. The algorithm first orients the edges using a low out-degree ordering, so each node only considers neighbors with higher order. Randomized Response is then applied to the original adjacency list, and the resulting noisy edges are used in combination with the oriented edges to count triangles. As shown in the figure, while the blue node is part of two true triangles in the original graph, only one triangle is preserved under the noisy edges.*

4.2 Theoretical Analysis

Memory Analysis & Communication Cost. Let M be the number of workers and n the graph size. Each worker processes S nodes, where $S = \lfloor n/M \rfloor$ for $M - 1$ workers, and the last worker handles $n - (M-1)\lfloor n/M \rfloor$ nodes. The coordinator manages three communication channels and publishes the noisy edges for the entire graph. The *cRR* structure, which aggregates noisy edges, requires $O(n^2)$ space, while *cTCount* and *cMaxOut*, which collect triangle counts and maximum noisy out-degree, require $O(M)$ space each. Storing published noisy edges further adds $O(n^2)$ space, resulting in a total coordinator memory requirement of $O(n^2 + M)$. Each worker processes $O(S)$ nodes, requiring $O(Sn)$ space for the graph. The *neighborsRR* structure for storing noisy edges demands $O(Sn)$ space, while computing the maximum noisy out-degree requires $O(S)$. The final triangle count computation takes $O(S \cdot \widetilde{d}_{\max})$ space, where $\widetilde{d}_{\max}$ is the maximum noisy out-degree, leading to an overall worker memory requirement of $O(Sn)$. The algorithm runs three communication rounds beyond those for low out-degree ordering. Workers first send noisy edges, incurring $O(Sn)$ communication cost, followed by sending the maximum noisy out-degree and triangle counts, each requiring $O(M)$ communication. Thus, the total communication overhead for the algorithm is $O(n^2 + M)$.

Privacy Guarantees. As before, our privacy guarantees are proven by implementing our triangle counting algorithm using local randomizers.

Lemma 4.2. *Our triangle counting algorithm is ε-LEDP.*

Proof. Our triangle counting algorithm calls Algorithm 3.1, which by Theorem 3.3 is $(\varepsilon/4)$-LEDP. Additionally, we release three sets of information, each of which we show to be $(\varepsilon/4)$-LEDP.

First, each node applies Randomized Response to the upper triangular adjacency matrix to generate a privacy-preserving set of edges. By [23], this adjacency list output is a $(\varepsilon/4)$-local randomizer.

Second, each node releases its privacy-preserving out-degree. By Line 7, the sensitivity of the out-degree (conditioning on Z) is 1 for neighboring adjacency lists. By the privacy of the geometric mechanism ([6, 13, 23, 24]), each node uses a $(\varepsilon/4)$-local randomizer to output its noisy degree.

Third, each node releases a privacy-preserving triangle count using its outgoing edges from the low out-degree ordering. To bound the sensitivity, we truncate the outgoing adjacency list (computed using Z) of each node by $\widetilde{d}_{\max}$. Given neighboring adjacency lists $\mathbf{a}$ and $\mathbf{a}'$, assume $\mathbf{a}'$ contains one additional neighbor w (without loss of generality). Let $\overline{\mathbf{a}}$ and $\overline{\mathbf{a}'}$ be the truncated adjacency lists. In the worst case, $\overline{\mathbf{a}}$ contains a node u not in $\overline{\mathbf{a}'}$, while $\overline{\mathbf{a}'}$ contains w (not in $\overline{\mathbf{a}}$). Let j be defined as in Line 9. If $j = u$, the first for-loop (Line 7) counts at most $\widetilde{d}_{\max}$ additional triangles for u (symmetrically for w). Assuming u returns $\widetilde{d}_{\max}$ triangles and w returns none, then for all other nodes $j \neq u$, the second for-loop (Line 8) encounters at most $\widetilde{d}_{\max}$ additional triangles. Thus, the total difference in counted triangles between $\overline{\mathbf{a}}$ and $\overline{\mathbf{a}'}$ is $2\widetilde{d}_{\max}$, giving a sensitivity of $2\widetilde{d}_{\max}$. By the privacy of the Laplace mechanism ([23]), outputting local triangle counts is an $(\varepsilon/4)$-local randomizer.

Since the differing edge between neighboring graphs G and G' affects at most one node's out-degree, applying composition ([22, 23, 25]) over all four $(\varepsilon/4)$-local randomizers results in an ε-LEDP triangle counting algorithm. □

Approximation Guarantees. One of the major novelties in our proofs is via a new intricate use of the Law of Total Expectation and Law of Total Variance for the events where the out-degrees of each node is upper bounded by the noisy maximum out-degree $\widetilde{d}_{\max}$ (which is, in turn, upper bounded by the degeneracy $\widetilde{O}(d)$). Such use cases were unnecessarily in [27, 44] because they did not use oriented edges. We first upper bound the out-degree by $\widetilde{O}\left(\frac{d}{\varepsilon}\right)$.

Lemma 4.3. *Given a graph where edges are oriented according to Algorithm 4.4, the maximum out-degree of any node is at most $O\left(d + \frac{\log(D_{\max})\log^2(n)}{\varepsilon}\right)$.*

Proof. By Invariant 1 and Theorem 3.4 (in the supplementary materials), the out-degree of any node v is at most $(2+\eta)k(v) + O\left(\frac{\log(D_{\max})\log^2(n)}{\varepsilon}\right)$, with high probability, where $k(v)$ is the core number of v. The largest core number is equal to the degeneracy of the graph [77]. Hence, the maximum out-degree of any node is upper bounded by $(2+\eta)d + O\left(\frac{\log(D_{\max})\log^2(n)}{\varepsilon}\right) = O\left(d + \frac{\log(D_{\max})\log^2(n)}{\varepsilon}\right)$ □

Lemma 4.4. *Given a graph where edges are oriented according to Algorithm 4.4, the number of oriented 4-cycles, denoted by $\overrightarrow{C}_4$, where each cycle contains two non-adjacent nodes with outgoing edges to the remaining two nodes (see Fig. 5), is at most $\widetilde{O}\left(\frac{n^2 d^2}{\varepsilon^2}\right)$, with high probability, where d is the degeneracy of the graph.*

Proof. There are $O(n^2)$ unordered pairs of vertices $\{w, x\}$ that may serve as the black nodes in an oriented 4-cycle (see Fig. 5). For a fixed pair $\{w, x\}$, define $S_{w,x} = \{u \in V : w \to u \text{ and } x \to u\}$ to be the set of vertices that are the outgoing endpoints of the outgoing edges from both w and x.

Under the low out-degree orientation computed by Algorithm 4.4 and by Lemma 4.3, each vertex has at most $O\left(d + \frac{\log(D_{\max})\log^2(n)}{\varepsilon}\right)$ out-neighbors and thus can have at most

[6] We use Laplace noise here as it offers a smoother tradeoff for smaller parameters.

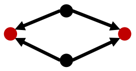


Figure 5: Oriented cycle of length 4; two non-adjacent black nodes have edges oriented toward the remaining red nodes.

$O\left(\left(d + \frac{\log(D_{\max})\log^2(n)}{\varepsilon}\right)^2\right) = O\left(d^2 + \frac{\log^2(D_{\max})\log^4(n)}{\varepsilon^2}\right)$ outgoing red pairs. Each pair $\{u, v\} \subseteq S_{w,x}$ forms an oriented 4-cycle with $\{w, x\}$, contributing at most this many cycles per black pair of vertices. Summing over all $O(n^2)$ black vertex pairs yields a total of at most $O\left(n^2\left(d^2 + \frac{\log^2(D_{\max})\log^4(n)}{\varepsilon^2}\right)\right) = \widetilde{O}\left(\frac{n^2d^2}{\varepsilon^2}\right)$ oriented 4-cycles. □

Lemma 4.5. *In expectation, our algorithm returns a 2-approximation of the true triangle count:* $\frac{(n^3-1)\cdot T}{n^3} \leq \mathbb{E}[\widetilde{\Delta}] \leq T$.

Proof. We first prove that, in expectation, $\mathbb{E}\left[\frac{X_{j,k}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1}\right] = \mathbb{1}_{j,k}$ where $\mathbb{1}_{j,k}$ is the indicator variable for whether edge $\{j, k\}$ exists in the original private graph. Since randomized response flips the bit indicating the existence of an edge between each pair of vertices with probability $\frac{1}{e^{\varepsilon/4}-1}$, we can simplify the expression to be:

$$\mathbb{E}\left[\frac{X_{j,k}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1}\right] = \frac{\mathbb{E}[X_{j,k}]\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1}.$$

When $\mathbb{1}_{j,k} = 1$, the probability that we obtain a bit of 1 is $\frac{e^{\varepsilon/4}}{e^{\varepsilon/4}+1}$ and our expression simplifies to

$$\frac{\mathbb{E}[X_{j,k}]\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1} = \frac{\frac{e^{\varepsilon/4}}{e^{\varepsilon/4}+1}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1} = 1.$$

When $\mathbb{1}_{j,k} = 0$, the probability that we obtain a bit of 1 is $\frac{1}{e^{\varepsilon/4}+1}$ then our expression simplifies to

$$\frac{\mathbb{E}[X_{j,k}]\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1} = \frac{\frac{1}{e^{\varepsilon/4}+1}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1} = 0.$$

The expected value of the random variable obtained for each edge e using our randomized response procedure is equal to $\mathbb{1}_e$.

Now, we condition on the event that $\widetilde{d}_{\max}$ upper bounds the out-degree of every vertex after computing and conditioning on D (the ordering). Let U be this event.

Then, for each node, we compute all pairs of its outgoing neighbors and use the random variable indicating existence of the edge spanned by the endpoints to count every triangle composed of a pair of outgoing edges. Let $T_{v,j,k}$ be the random variable representing the existence of the queried triangle for node v and outgoing edges (v, j) and (v, k). Then, $\mathbb{E}[T_{v,j,k} \mid U] = \mathbb{E}[X_{j,k}]$ since both outgoing edges (v, j) and (v, k) exists. By what we showed above, it then follows that $\mathbb{E}[T_{v,j,k}] = \mathbb{1}_{j,k}$.

We must account for the symmetric geometric noise. Since the expectation of the symmetric geometric noise is 0 and we add together all of the values for each of the drawn noises, the expected total noise is 0 by linearity of expectations. Each triangle has a unique node that queries it since every triangle has a unique orientation of edges where two edges are outgoing from a vertex in the triangle. Hence, the expected sum of all queried triangles is T, conditioned on U, by linearity of expectations.

Finally, we remove the conditioning on the event U. Recall that $\mathbb{E}[W] = \sum_y \mathbb{E}[W \mid Y = y] \cdot \Pr(Y = y)$. Since truncation can only decrease the number of queried triangles (and hence the expectation), we upper and lower bound $0 \leq \mathbb{E}[\widetilde{\Delta} \mid \neg U] \leq T$. Hence we only need to figure out the probability $P(U)$ to upper and lower bound $\mathbb{E}[\widetilde{\Delta}]$. The probability of event U is the probability that $\max(\{d_v + \mathrm{Geom}(\varepsilon/4) \mid v \in V\}) + \frac{\log(n)}{\varepsilon} \geq \max(\{d_v \mid v \in V\})$ where d_v is the out-degree of node v given order D. This probability is lower bounded by the probability that $\max(\{d_v \mid v \in V\}) + \mathrm{Geom}(\varepsilon/4) + \frac{c\log(n)}{\varepsilon} \geq \max(\{d_v \mid v \in V\})$ for a fixed constant $c \geq 1$; this means we want to lower bound $\Pr\left(\left(Q + \frac{c\log(n)}{\varepsilon}\right) \geq 0\right)$ where $Q \sim \mathrm{Geom}(\varepsilon/4)$. By concentration of random variables chosen from the symmetrical geometric distribution, we know that $\Pr\left(\left(Q + \frac{c\log(n)}{\varepsilon}\right) \geq 0\right) \geq 1 - \frac{1}{n^3}$ for large enough constant $c \geq 1$. Specifically, setting $c = 3$ gives us this bound. Hence, we can upper and lower bound $\frac{(n^3-1)\cdot T}{n^3} \leq \mathbb{E}[\widetilde{\Delta}] \leq T$. □

To calculate the variance of the triangle count obtained from our algorithm, we use a quantity denoted by $\overrightarrow{C}_4$ which is the number of oriented 4-cycles where there exists two non-adjacent nodes with outgoing edges to the other two nodes in the cycle. See Fig. 5 for an example. The number of oriented cycles of length 4 is upper bounded by n^2d^2 where d is the degeneracy (maximum core number) in the graph, resulting in significant gains in utility over previous results which use the number of total (not oriented) 4-cycles which could be as large as $\Omega(n^4)$.

Lemma 4.6. *Our triangle counting algorithm returns a count with variance* $O\left(\frac{nd^2\log^6 n}{\varepsilon^4} + \overrightarrow{C}_4\right)$ *where d is the degeneracy (max core number) of the graph,* $\overrightarrow{C}_4$ *is the number of oriented cycles of length 4, and T is the number of (true) triangles in the private graph.*

Proof. As before, we first calculate the variance conditioned on the event U that $\widetilde{d}_{\max}$ upper bounds the out-degrees of every node. Then, we use the law of total variance to remove this condition.

For notation simplicity, we omit the condition on U from the right hand sides of the below equations. First, the variance of $\mathrm{Var}[X_{\{i,j\}}] = \frac{e^{\varepsilon}}{(e^{\varepsilon}+1)^2}$, since $X_{\{i,j\}}$ is a Bernoulli variable. Then, let $\hat{T}$ be our returned triangle count. The variance

$$\begin{aligned}&\mathrm{Var}\left[\hat{T} \mid U\right]\\&= \mathrm{Var}\left[\sum_{v\in[n]}\left(\sum_{j,k\in Out(v)}\left(\frac{X_{j,k}\cdot(e^{\varepsilon/4}+1)-1}{e^{\varepsilon/4}-1}\right) + \mathrm{Lap}\left(\frac{\varepsilon}{2\widetilde{d}_{\max}}\right)\right)\right].\end{aligned}$$

Recall $\widetilde{d}_{\max}$ is our noisy maximum out-degree of any vertex. Then, our variance simplifies to

$$\left(\frac{e^{\varepsilon/4}+1}{e^{\varepsilon/4}-1}\right)^2 \cdot \mathrm{Var}\left[\sum_{v\in[n]}\sum_{j,k\in Out(v)} X_{j,k}\right] + \mathrm{Var}\left[\sum_{v\in[n]}\mathrm{Lap}\left(\frac{\varepsilon}{2\widetilde{d}_{\max}}\right)\right].$$

By the variance of the Laplace distribution, we can compute

$$\mathrm{Var}\left[\sum_{v\in[n]}\mathrm{Lap}\left(\frac{\varepsilon}{2\widetilde{d}_{\max}}\right)\right] \leq n \cdot \frac{8\widetilde{d}^2_{\max}}{\varepsilon^2}.$$

Now, it remains to compute $\mathrm{Var}\left[\sum_{v\in[n]}\sum_{j,k\in Out(v)} X_{j,k}\right]$. The covariance is 0 if two queried pairs do not query the same $X_{j,k}$. The covariance is non-zero only in the case of queries which overlap in $X_{i,j}$. This occurs only in the case of an oriented 4-cycle where $X_{i,j}$ is shared between two queries. Let P_2 be the set of all pairs of such queries that share $X_{j,k}$. In this case, the covariance is upper bounded by $\mathbb{E}[X_{j,k}^2]$. Hence, we can simplify our expression to be

$$\mathrm{Var}\left[\sum_{v\in[n]}\sum_{j,k\in Out(v)} X_{j,k}\right] \tag{1}$$

$$\leq \sum_{v\in[n]}\sum_{j,k\in Out(v)} \mathrm{Var}\left[X_{j,k}\right] + 2\cdot \sum_{T_{v,j,k},T_{w,j,k}\in P_2} \left(\mathbb{E}[X_{j,k}^2]\right) \tag{2}$$

$$\leq n\cdot \widetilde{d}_{\max}^2 \cdot \frac{e^{\varepsilon/4}}{(e^{\varepsilon/4}+1)^2} + \overrightarrow{C}_4 \cdot \left(\frac{e^{\varepsilon/4}}{e^{\varepsilon/4}+1}\right), \tag{3}$$

where $\overrightarrow{C}_4$ indicates the number of directed cycle of length 4 (Fig. 5).

Hence, our total variance is upper bounded by

$$\frac{8n\widetilde{d}_{\max}^2}{\varepsilon^2} + n\cdot \widetilde{d}_{\max}^2 \cdot \frac{e^{\varepsilon/4}}{(e^{\varepsilon/4}+1)^2} + \overrightarrow{C}_4 \cdot \frac{e^{\varepsilon/4}}{e^{\varepsilon/4}+1}.$$

Finally, by the guarantees of our k-core decomposition algorithm, the maximum out-degree $d_{\max}$ is bounded by $d_{\max} \leq (2+\eta)\cdot d + O\left(\frac{\log(D_{\max})\log^2 n}{\varepsilon}\right)$, with high probability, where d is the degeneracy of the input graph and $D_{\max} \leq n$ is the maximum degree in the graph. Finally, we also know that the noise generated for $\widetilde{d}_{\max}$ is upper bounded by $O\left(\frac{\log n}{\varepsilon}\right)$. Thus, $\widetilde{d}_{\max} \leq (2+\eta)\cdot d + O\left(\frac{\log^3 n}{\varepsilon}\right)$ Hence, our variance is upper bounded by $O\left(\frac{nd^2\log^6 n}{\varepsilon^4} + \overrightarrow{C}_4\right)$.

We now remove our condition on U and use the law of total variance to compute our unconditional variance. Recall the law of total variance states $\mathrm{Var}[Y] = \mathbb{E}[\mathrm{Var}[Y \mid X]] + \mathrm{Var}[\mathbb{E}[Y \mid X]]$. We computed $\mathrm{Var}\left[\hat{T} \mid U\right]$ above. We now compute $\mathrm{Var}\left[\hat{T} \mid \neg U\right]$. The main difference between when the event U occurs and does not occur is that some adjacency lists of the outgoing neighbors will be truncated. Consequently, we sum over fewer $X_{j,k}$ variables in Eqs. (1) and (2). Thus, the variance when U does not occur is upper bounded by the variance when U does occur.

Now we calculate $\mathrm{Var}\left[\mathbb{E}[\hat{T} \mid U]\right] = \mathbb{E}[\mathbb{E}[\hat{T} \mid U]^2] - \mathbb{E}[\mathbb{E}[\hat{T} \mid U]]^2$. By our calculation in the proof of Lemma 4.5, we can calculate $\mathbb{E}[\hat{T} \mid U] = T$ and let $\frac{(n^3-1)\cdot T}{n^3} \leq Y = \mathbb{E}[\hat{T} \mid \neg U] \leq T$. Then,

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[\hat{T} \mid U]^2] - \mathbb{E}[\mathbb{E}[\hat{T} \mid U]]^2 &= \left(\frac{T^2}{2} + \frac{Y^2}{2}\right) - \left(\frac{T}{2} + \frac{Y}{2}\right)^2 \\
&= \frac{T^2+Y^2}{2} - \frac{T^2+2TY+Y^2}{4} \\
&= \frac{T^2+Y^2-2TY}{4} \\
&= \left(\frac{T-Y}{2}\right)^2 \\
&\leq \left(\frac{T - \frac{(n^3-1)\cdot T}{n^3}}{2}\right)^2 \\
&\leq \left(\frac{T}{2n^3}\right)^2 \\
&\leq \frac{1}{4}.
\end{aligned}$$

Thus, our final variance is $O\left(\frac{nd^2\log^6 n}{\varepsilon^4} + \overrightarrow{C}_4\right)$. □

Theorem 4.7. *With high constant probability, our triangle counting algorithm returns a* $\left(1+\eta, O\left(\frac{\sqrt{n}d\log^3 n}{\varepsilon^2} + \sqrt{\overrightarrow{C}_4}\right)\right)$*-approximation of the true triangle count.*

Proof. We use Chebyshev's inequality with the standard deviation calculated from Lemma 4.6. The $(1+\eta)$-approximation comes from our $\left(1-\frac{1}{n^3}\right)$-approximation of the expectation. □

See Table 2 for the d values of real-world graphs; when $d = O(1)$ is constant, as is the case for real-world graphs, then $\overrightarrow{C}_4 = O(n^2)$ and $T = O(n)$. We improve previous theoretical additive errors from $O\left(\frac{\sqrt{C_4}}{\varepsilon} + \frac{n^{3/2}}{\varepsilon^2}\right)$ [44] to $O\left(\sqrt{\overrightarrow{C}_4} + \frac{\sqrt{n}\log^3 n}{\varepsilon^2}\right)$, an improvement of at least a $\Omega(\sqrt{n})$ factor, translating to massive practical gains.

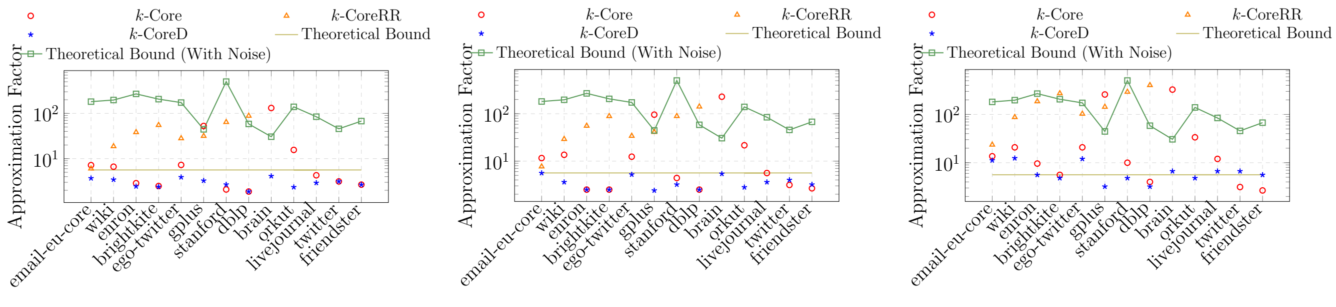
5 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance and accuracy of our k-core decomposition (k-CoreD) and triangle counting (EdgeOrient$_\Delta$) algorithms under Local Edge Differential Privacy (LEDP) using a distributed simulation. We benchmark against prior LEDP algorithms, and to highlight the limitations of Randomized Response (RR), we additionally implement RR-based baselines for both problems. All RR baselines are evaluated purely in terms of accuracy, as they are centralized algorithms and not directly comparable in runtime to our distributed setting. Furthermore, RR introduces significant computational overhead due to increased graph density from edge perturbation. **Consequently, we omit RR results on many large graphs: k-core baselines fail due to out-of-memory (OOM) errors, while triangle counting exceeds timeout limits. These failures occur independently, underscoring the instability and inefficiency of naive RR methods on large-scale graphs.**

k-Core Baselines. We compare k-CoreD against the LEDP k-core decomposition algorithm of [19] (denoted k-Core), which we implement. Additionally, we construct an RR-based baseline (denoted k-CoreRR) that runs the standard peeling algorithm on the RR-perturbed graph and applies a scaling factor to correct the induced degrees, ensuring unbiased estimates. Our method achieves significantly better accuracy, reducing approximation error by up to **two orders of magnitude** over k-CoreRR and outperforming k-Core [19] in both accuracy and efficiency—reducing the number of rounds by nearly **two orders of magnitude** across all graphs. k-CoreRR consistently fails on larger datasets due to memory exhaustion caused by increased graph density.

Triangle Counting Baselines. We compare EdgeOrient$_\Delta$ against two LEDP triangle counting algorithms: **ARROneNS$_\Delta$ (Lap)**[44] and **GroupRR**[39]. For the RR baseline (denoted as TCountRR), we follow the approach from [26]. Our algorithm achieves up to **six orders of magnitude** improvement in multiplicative accuracy while providing substantial speedups in larger graphs.

Experimental Setup. To evaluate our algorithms in a distributed simulation, we partition the input graph across M **worker processors** and a **single coordinator processor**. Each worker handles a subset of nodes and their full adjacency lists, running LEDP algorithms locally. Workers communicate their privacy-preserving



(a) Average Approximation Factor

(b) 80^{th} Percentile Approximation Factor

(c) 95^{th} Percentile Approximation Factor

Figure 6: k-core Decomposition Results.

Table 2: Graph size, maximum core number, and number of triangles.

Graph Name	Num. Vertices	Num. Edges	Max. Degree	Max. Core Num. (d)	Num. Triangles
email-eu-core	986	1,329,336	345	34	105,461
wiki	7115	100,761	1065	35	608,387
enron	36,692	183,830	1,383	43	727,044
brightkite	58,228	214,078	1,134	52	494,728
ego-twitter	81,306	1,342,296	3,383	96	13,082,506
gplus	107,614	12,238,285	20,127	752	1,073,677,742
stanford	281,903	1,992,635	38,625	71	11,329,473
dblp	317,080	1,049,866	343	113	2,224,385
brain	784,262	267,844,669	21,743	1200	–
orkut	3,072,441	117,185,083	33,313	253	–
livejournal	4,846,609	42,851,237	20,333	372	–
twitter	41,652,230	1,202,513,046	2,997,487	2488	–
friendster	65,608,366	1,806,067,135	5214	304	–

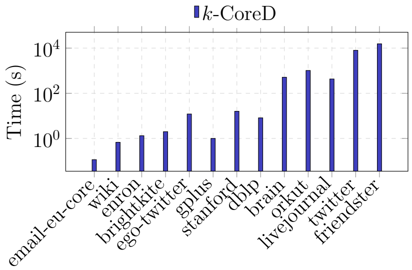


Figure 7: k-Core Decomposition Avg. Response Time

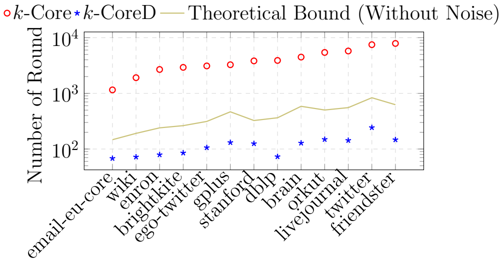


Figure 8: k-Core Decomposition Number of Rounds

outputs to the coordinator, which aggregates the data and broadcasts new public information. This proceeds over multiple synchronous rounds, simulating a real-world distributed setting. We use **80 worker processors** and **a single coordinator**. Each value is averaged over **5 runs**, with a **4-hour** wall-clock limit per run.

Parameters We use $\varepsilon = 1.0$, bias term = 8, approximation factor $(2 + \eta) = 5.625$ (matching non-private k-core experiments [55]), and privacy split fraction $f = 0.8$ (allocating $0.8 \cdot \varepsilon$ to thresholding and $0.2 \cdot \varepsilon$ to level moving step). We also conduct an ablation study on key parameters ε and f, and our theoretical proofs show the approximation falls within a $(2 + \eta)$-multiplicative factor.

Compute Resources We run experiments on a Google Cloud `c3-standard-176` instance (3.3 GHz Intel Sapphire Rapids CPUs, 88 physical cores, 704 GiB RAM) with hyper-threading disabled. The code, implemented in Golang [21], is publicly available [1].

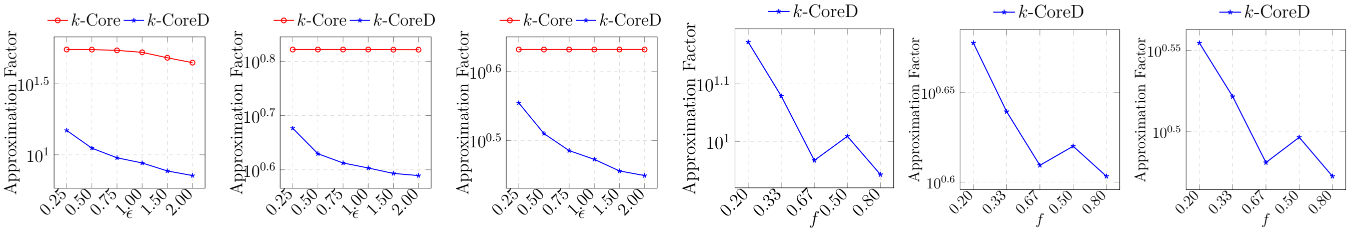
Datasets We test our algorithms on a diverse set of 13 real-world undirected graphs from SNAP [51], the DIMACS Shortest Paths Challenge road networks [16], and the Network Repository [74]: ***email-eu-core***, ***wiki***, ***enron***, ***brightkite***, ***ego-twitter***, ***gplus***, ***stanford***, ***dblp***, ***orkut***, ***livejournal***, and ***friendster***. We also use ***twitter***, a symmetrized version of the Twitter network [50], and ***brain***, a highly dense human brain network from NeuroData (https://neurodata.io/). We remove duplicate edges, zero-degree vertices, and self-loops. Table 2 reflects the graph statistics after this removal. Exact triangle counts for some graphs are omitted due to time or memory constraints.

5.1 k-Core Decomposition

Response Time. Fig. 7 shows the response times of k-CoreD across all datasets. Our algorithm efficiently processes large-scale graphs, including billion-edge datasets like *twitter* and *friendster*, within **four hours**. These results validate the scalability and practicality of k-CoreD, demonstrating the impact of degree thresholding and bias terms in delivering both theoretical and practical improvements.

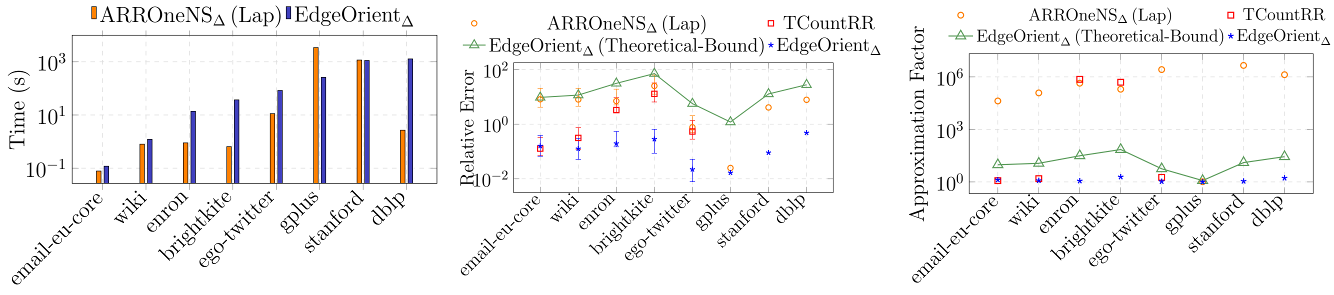
While measuring response time, a direct runtime comparison with the k-Core algorithm [19] is not meaningful. That algorithm does not include bias correction, which often results in large negative noise causing most nodes to remain stuck at level 0. Since the algorithm proceeds for a fixed number of communication rounds (matching the levels in the LDS), but no nodes move beyond the first level, negligible work is performed in subsequent rounds—making the runtime unrealistically low while returning (the same) poor approximation for many nodes. Instead, we compare the number of communication rounds. As shown in Fig. 8, k-CoreD reduces the number of rounds by ***two orders of magnitude*** compared to the baseline, aligning with our theoretical bound of $O(\log(n) \cdot \log(D_{\max}))$. This translates to significantly lower communication overhead and improved scalability in distributed settings.

Accuracy We calculate the approximation factor for each node as $a_v = \frac{\max(s_v,t_v)}{\min(s_v,t_v)}$, where s_v is the approximate core number and t_v is the true core number. We use this metric to be consistent with the best-known *non-private* k-core decomposition implementations [17, 55]. These individual node approximation factors facilitate the computation of aggregate metrics: the average, maximum, 80^{th}, and 95^{th} percentile approximation factors for each graph. The theoretical approximation bound in the absence of noise is calculated as $(2 + \eta)$, which is 5.625 for all graphs (labeled Theoretical



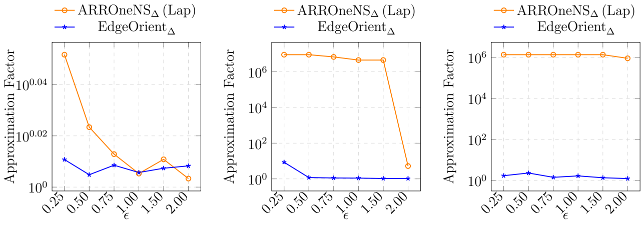
(a) gplus (b) wiki (c) livejournal (d) gplus (e) wiki (f) livejournal

Figure 9: Avg. approximation factor for k-Core decomposition vs. epsilon (ε) and split fraction (f).



(a) Average Response Time (b) Relative Error (c) Average Approximation Factor

Figure 10: Triangle Counting Results.



(a) gplus (b) stanford (c) dblp

Figure 11: Avg. approx factor for triangle counting vs. epsilon (ε).

Bound). Additionally, we adjust this bound to account for noise by incorporating the additive error term $\frac{\log^3_{1+\eta/5}(D_{max})}{\varepsilon}$, where n is the number of nodes, D_{max} is the maximum degree (labeled Theoretical Bound (With Noise)). For this bound, we compute the effect of the additive noise on the multiplicative factor by adding $\frac{\log^3_{1+\eta/5}(D_{max})}{\varepsilon \cdot k_{\max}}$, where $k_{\max}$ is the maximum core number, to 5.625. Note that such a theoretical bound is a *lower bound* on the effect of the additive error on the multiplicative factor; for smaller core numbers, e.g. $k_{\min} << k_{\max}$, the additive error leads to a *much greater* factor.

Figs. 6a to 6c present the approximation factors achieved by our approach (k-CoreD) compared to the baseline k-core algorithm (k-Core) from [19], and the randomized response baseline (k-CoreRR), along with theoretical bounds across various datasets. On average, our method maintains approximation factors below **4x** across all datasets, with the 80th percentile staying under **5.5x**, as illustrated in Fig. 6a and Fig. 6b, demonstrating significantly lower variance compared to the baselines. These results remain well within the theoretical bounds without noise. Compared to k-Core [19], k-CoreD consistently achieves better or comparable performance. Notably, for graphs such as *brain* and *gplus*, k-CoreD reduces the approximation factors from 131.55 to 4.11 and from 52.71 to 3.27, improvements of over **31x** and **16x**, respectively. Similarly, for *orkut* and *wiki*, our algorithm improves the approximation factors by **6.6x** and **1.9x**, respectively. Compared to k-CoreRR, our algorithm achieves consistently better performance across all datasets—often by **two to three orders of magnitude**. Notably, on the *dblp* graph, k-CoreD improves the approximation factor by over **47x**. These results underscore the limitations of naive RR-based methods, which suffer from inflated graph density due to edge perturbation.

However, for many graphs, the difference between k-Core [19] and our approach is less pronounced. This is due to the k-Core algorithm's inability to move nodes up levels in the LDS, which results in an approximate core number of 2.5 for most nodes. Given that real-world graphs often exhibit small core numbers (Table 2), the average approximation factor for k-Core becomes skewed, particularly for graphs with a significant proportion of low-core nodes. In contrast, for graphs with larger core numbers, such as *gplus*, *brain*, and *orkut*, our method demonstrates significant improvements. The advantages of our algorithm become more evident when examining the 80th and 95th percentile approximation factors. As shown in Fig. 6b, our method consistently achieves a notable reduction in the 80th percentile error compared to k-Core, with reductions of up to **42x**, **40x**, and **7.7x** for graphs with large core numbers, such as *brain*, *gplus*, and *orkut*, respectively. Similarly, in the 95th percentile error (Fig. 6c), our method achieves reductions of nearly **49x**, **81x**, and **7x** for the same graphs. These results underscore the robustness and scalability of our algorithm in handling diverse graph structures with varying core number distributions.

Ablation Study We analyze the effect of varying the privacy parameter, ε, and the privacy split fraction, f, on the utility of the k-core decomposition algorithm by plotting the average approximation factor across different datasets: *gplus*, *wiki*, and *livejournal*, for our algorithm and the baseline. From the results shown in Fig. 9, we observe that the approximation factor improves as ε increases, which aligns with the theoretical expectations of differential privacy where higher ε allows for less noise and greater utility.

Further, we note that an optimal value of 0.8 consistently minimizes the approximation factor across all datasets (Fig. 9). This is the case since degree thresholding affects the amount of noise that will be added per level for later computations. Thus, we use $f = 0.8$ for all other experiments, as it strikes a balance between the two steps, ensuring better overall performance.

5.2 Triangle Counting

Response Time Fig. 10a shows the average response times of our LEDP triangle counting algorithm, EdgeOrient$_\Delta$, implemented in our distributed (LEDP-DS) framework. We compare our algorithm to **ARROneNS$_\Delta$ (Lap)** from [44]. While ARROneNS$_\Delta$ (Lap) is implemented in C++, our Golang implementation demonstrates comparable performance across most datasets, with notable speedups for large graphs. Specifically, for *gplus*, our algorithm achieves a **speedup of 3.45x**, while for other graphs such as *email-eu-core, stanford*, and *wiki*, our performance is comparable despite the communication overhead. However, for the *enron, brightkite*, and *dblp* dataset, our algorithm is slower. This discrepancy is likely due to the smaller sizes of the graphs so a centralized algorithm will perform better than a distributed algorithm. These results emphasize the scalability and practicality of EdgeOrient$_\Delta$ for large-scale graph analysis, highlighting its ability to handle diverse graphs.

Accuracy Following the evaluation methodology in [44], we compute relative error for a graph using $\frac{|\widetilde{\Delta}-\Delta|}{\Delta}$, where $\widetilde{\Delta}$ represents the approximated triangle count and Δ the true triangle count. Additionally, we apply the theoretical bounds from Theorem 4.7 to our analysis. According to Fig. 10b, our algorithm, EdgeOrient$_\Delta$, consistently achieves relative errors ranging from $\mathbf{10^{-1}}$ to $\mathbf{10^{-2}}$ across all datasets, remaining well within the theoretical bounds.

Compared to ARROneNS$_\Delta$ (Lap), our algorithm gives better relative errors by **53x - 89x**, for all graphs except *gplus*, where we achieve slightly better but comparable accuracy. In contrast to GroupRR [39], which we could only run on the *wiki* dataset due to the 4-hour timeout limit, our algorithm not only matches accuracy but also has a response time ***two orders of magnitude faster***.

On small or dense graphs like *wiki* or *email-eu-core*, the TCountRR—which incurs a dominant error term growing proportional to $\frac{n^{3/2}}{\varepsilon^2}$—performs similarly to ours because n is small [26]. However, in larger graphs, $n^{3/2}$ explodes, whereas our error depends instead on $\sqrt{n}\, d$ (with $d \ll n$). TCountRR must also consider all $O(n^4)$ 4-cycles, but our method only needs $\widetilde{O}(n^2 d^2)$ 4-cycles. Consequently, on larger datasets such as *enron* and *brightkite*, we reduce the relative error (Fig. 10b) by roughly **17x** and **45.5x**, respectively. Compared to using 60 worker cores, we achieve up to a 1.41x speedup.

To further analyze the limitations of prior approaches, we compute the multiplicative approximation factor of the triangle count, defined as $\frac{\max\left(\widetilde{\Delta},\Delta\right)}{\max\left(1,\min\left(\widetilde{\Delta},\Delta\right)\right)}$. Unlike relative error, this metric explicitly accounts for cases where algorithms, such as the one in [44], produce negative triangle counts, which severely undermines their utility in real-world scenarios. As shown in Fig. 10c, our algorithm achieves consistently small approximation factors across all graphs, remaining within $[\mathbf{1.01}, \mathbf{1.93}]$, and reducing the factor by **six orders of magnitude** compared to [44] and TCountRR. This highlights the robustness of our approach, EdgeOrient$_\Delta$, in maintaining low and stable approximation factors across diverse graph structures.

Ablation Study To evaluate the impact of the privacy parameter ε on utility, we analyze and plot the approximation factors for varying values of ε across three representative graphs: *gplus, dblp*, and *stanford*, comparing against those reported in [44]. Our results demonstrate that for *stanford* and *dblp*, our algorithm achieves a significant reduction in approximation factor by up to **six orders of magnitude**. For *gplus*, while the approximation factors are comparable for higher values of ε, our algorithm achieves better utility for smaller values of ε. Specifically, at $\varepsilon = 0.25$, our algorithm achieves an approximation factor of 1.025, compared to 1.126 for ARROneNS$_\Delta$ (Lap), an improvement of **9.8%**. Similarly, at $\varepsilon = 0.50$, our algorithm achieves a factor of 1.011, compared to 1.055, an improvement of **4.2%** This highlights the ability of our approach to offer better utility even under stricter privacy constraints, underscoring its advantage over [44]. Additionally, we observe that the utility of our algorithm consistently improves as the privacy parameter ε increases, aligning with the theoretical expectations of differential privacy, where higher values of ε results in less noise.

6 CONCLUSION

Large-scale network analysis often raises privacy concerns for sensitive data. We employ local edge differential privacy (LEDP), letting nodes protect their edges without a trusted authority. We propose novel LEDP algorithms for k-core decomposition and triangle counting that surpass prior work in accuracy and theoretical guarantees and introduce the first distributed framework to simulate these algorithms on a single machine. Experiments show our k-core approximations meet non-private theoretical bounds on average, while triangle counting errors are nearly two orders of magnitude lower than previous LEDP methods, with similar runtimes. Ongoing work extends this framework to multi-machine testing with real communication to capture added latency and explores multi-coordinator variants to eliminate single points of failure and reduce bottlenecks. These enhancements will support a broader range of LEDP algorithms. Our open-source framework (at [1]) invites the community to build on LEDP graph algorithms.

REFERENCES

[1] 2024. DistributedLEDPGraphAlgos. https://github.com/mundrapranay/DistributedLEDPGraphAlgos.

[2] John M Abowd, Robert Ashmead, Ryan Cumings-Menon, Simson Garfinkel, Micah Heineck, Christine Heiss, Robert Johns, Daniel Kifer, Philip Leclerc, Ashwin Machanavajjhala, et al. 2022. The 2020 census disclosure avoidance system topdown algorithm. *Harvard Data Science Review* 2 (2022).

[3] Jun Ai, Yayun Liu, Zhan Su, Fengyu Zhao, and Dunlu Peng. 2021. K-core decomposition in recommender systems improves accuracy of rating prediction. *International Journal of Modern Physics C* 32, 07 (2021), 2150087. https://doi.org/10.1142/S012918312150087X

[4] Mohammad Al Hasan and Vachik S Dave. 2018. Triangle counting in large networks: a review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 2 (2018), e1226.

[5] Noga Alon, Raphael Yuster, and Uri Zwick. 1997. Finding and counting given length cycles. *Algorithmica* 17, 3 (1997), 209–223.

[6] Victor Balcer and Salil P. Vadhan. 2018. Differential Privacy on Finite Computers. In *9th Innovations in Theoretical Computer Science Conference (ITCS)*. 43:1–43:21.

[7] Bradley R Bebee, Daniel Choi, Ankit Gupta, Andi Gutmans, Ankesh Khandelwal, Yigit Kiran, Sainath Mallidi, Bruce McGaughy, Mike Personick, Karthik Rajan, et al. 2018. Amazon Neptune: Graph Data Management in the Cloud.. In *ISWC (P&D/Industry/BlueSky)*.

[8] Arijit Bishnu, Debarshi Chanda, and Gopinath Mishra. 2025. Arboricity and Random Edge Queries Matter for Triangle Counting using Sublinear Queries. arXiv:2502.15379 [cs.DS] https://arxiv.org/abs/2502.15379

[9] Arijit Bishnu, Debarshi Chanda, and Gopinath Mishra. 2025. Arboricity and Random Edge Queries Matter for Triangle Counting using Sublinear Queries. *CoRR* abs/2502.15379 (February 2025). https://doi.org/10.48550/arXiv.2502.15379

[10] Francesco Bonchi, Aristides Gionis, and Francesco Gullo. 2014. Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1316–1325.

[11] Felipe T Brito, Victor AE Farias, Cheryl Flynn, Subhabrata Majumdar, Javam C Machado, and Divesh Srivastava. 2023. Global and local differentially private release of count-weighted graphs. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.

[12] Mark Bun and Thomas Steinke. 2016. Concentrated Differential Privacy: Simplifications, Extensions, and Lower Bounds. In *International Conference on Theory of Cryptography*. 635–658.

[13] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2011. Private and Continual Release of Statistics. *ACM Trans. Inf. Syst. Secur.* 14, 3, Article 26 (Nov. 2011), 24 pages. https://doi.org/10.1145/2043621.2043626

[14] Ho-Chun Herbert Chang and Emilio Ferrara. 2022. Comparative analysis of social bots and humans during the COVID-19 pandemic. *Journal of Computational Social Science* 5, 2 (2022), 1409–1425.

[15] Martino Ciaperoni, Edoardo Galimberti, Francesco Bonchi, Ciro Cattuto, Francesco Gullo, and Alain Barrat. 2020. Relevance of temporal cores for epidemic spread in temporal networks. *Scientific reports* 10, 1 (2020), 12529.

[16] Camil Demetrescu, Andrew V Goldberg, David S Johnson, et al. 2008. Implementation challenge for shortest paths. In *Encyclopedia of Algorithms*. Springer US, 395–398.

[17] Laxman Dhulipala, Guy E. Blelloch, and Julian Shun. 2017. Julienne: A Framework for Parallel Graph Algorithms Using Work-efficient Bucketing. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 293–304.

[18] Laxman Dhulipala, George Z. Li, and Quanquan C. Liu. 2024. Near-Optimal Differentially Private k-Core Decomposition. arXiv:2312.07706 [cs.DS] https://arxiv.org/abs/2312.07706

[19] 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 *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 754–765.

[20] Michael Dinitz, Satyen Kale, Silvio Lattanzi, and Sergei Vassilvitskii. 2024. Almost Tight Bounds for Differentially Private Densest Subgraph. arXiv:2308.10316 [cs.DS]

[21] Alan A.A. Donovan and Brian W. Kernighan. 2015. *The Go Programming Language* (1st ed.). Addison-Wesley Professional.

[22] Cynthia Dwork and Jing Lei. 2009. Differential Privacy and Robust Statistics. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. 371–380.

[23] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Third Conference on Theory of Cryptography*. 265–284.

[24] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential Privacy under Continual Observation. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*. 715–724.

[25] Cynthia Dwork, Guy N. Rothblum, and Salil Vadhan. 2010. Boosting and Differential Privacy. In *Proceedings of the IEEE 51st Annual Symposium on Foundations of Computer Science*. 51–60.

[26] Talya Eden, Quanquan C. Liu, Sofya Raskhodnikova, and Adam Smith. 2023. Triangle Counting with Local Edge Differential Privacy. In *50th International Colloquium on Automata, Languages, and Programming (ICALP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 261)*, Kousha Etessami, Uriel Feige, and Gabriele Puppis (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 52:1–52:21. https://doi.org/10.4230/LIPIcs.ICALP.2023.52

[27] Talya Eden, Quanquan C. Liu, Sofya Raskhodnikova, and Adam D. Smith. 2023. Triangle Counting with Local Edge Differential Privacy. In *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany (LIPIcs, Vol. 261)*, Kousha Etessami, Uriel Feige, and Gabriele Puppis (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 52:1–52:21. https://doi.org/10.4230/LIPICS.ICALP.2023.52

[28] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. 2003. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 211–222.

[29] Alireza Farhadi, MohammadTaghi Hajiaghayi, and Elaine Shi. 2021. Differentially Private Densest Subgraph. *CoRR* abs/2106.00508 (2021), 11581–11597. arXiv:2106.00508 https://arxiv.org/abs/2106.00508

[30] Victor AE Farias, Felipe T Brito, Cheryl Flynn, Javam C Machado, Subhabrata Majumdar, and Divesh Srivastava. 2020. Local dampening: Differential privacy for non-numeric queries via local sensitivity. *arXiv preprint arXiv:2012.04117* (2020).

[31] Nan Fu, Weiwei Ni, Sen Zhang, Lihe Hou, and Dongyue Zhang. 2023. GC-NLDP: A graph clustering algorithm with local differential privacy. *Computers & Security* 124 (2023), 102967.

[32] Kayo Fujimoto, Dimitrios Paraskevis, Jacky C Kuo, Camden J Hallmark, Jing Zhao, Andre Hochi, Lisa M Kuhns, Lu-Yu Hwang, Angelos Hatzakis, and John A Schneider. 2022. Integrated molecular and affiliation network analysis: Core-periphery social clustering is associated with HIV transmission patterns. *Social networks* 68 (2022), 107–117.

[33] Christos Giatsidis, Fragkiskos D Malliaros, Nikolaos Tziortziotis, Charanpal Dhanjal, Emmanouil Kiagias, Dimitrios M Thilikos, and Michalis Vazirgiannis. 2016. A k-core decomposition framework for graph clustering. *arXiv preprint arXiv:1607.02096* (2016).

[34] Taolin Guo, Shunshun Peng, Yong Li, Mingliang Zhou, and Trieu-Kien Truong. 2023. Community-based social recommendation under local differential privacy protection. *Information Sciences* 639 (2023), 119002.

[35] Yang Guo, Fatemeh Esfahani, Xiaojian Shao, Venkatesh Srinivasan, Alex Thomo, Li Xing, and Xuekui Zhang. 2022. Integrative COVID-19 biological network inference with probabilistic core decomposition. *Briefings in Bioinformatics* 23, 1 (2022), bbab455.

[36] Jonathan Hehir, Aleksandra Slavković, and Xiaoyue Niu. 2022. Consistent spectral clustering of network block models under local differential privacy. *The Journal of privacy and confidentiality* 12, 2 (2022).

[37] Monika Henzinger, A. R. Sricharan, and Leqi Zhu. 2024. Tighter Bounds for Local Differentially Private Core Decomposition and Densest Subgraph. *CoRR* abs/2402.18020 (2024). https://doi.org/10.48550/ARXIV.2402.18020 arXiv:2402.18020

[38] Seira Hidano and Takao Murakami. 2022. Degree-preserving randomized response for graph neural networks under local differential privacy. *arXiv preprint arXiv:2202.10209* (2022).

[39] 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* (2023).

[40] Quentin Hillebrand, Vorapong Suppakitpaisarn, and Tetsuo Shibuya. 2023. Unbiased locally private estimator for polynomials of laplacian variables. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 741–751.

[41] Petter Holme and Nelly Litvak. 2017. Cost-efficient vaccination protocols for network epidemiology. *PLoS computational biology* 13, 9 (2017), e1005696.

[42] Jacob Imola, Alessandro Epasto, Mohammad Mahdian, Vincent Cohen-Addad, and Vahab Mirrokni. 2023. Differentially private hierarchical clustering with provable approximation guarantees. In *International Conference on Machine Learning*. PMLR, 14353–14375.

[43] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2021. Locally Differentially Private Analysis of Graph Statistics. In *30th USENIX Security Symposium*. 983–1000.

[44] Jacob Imola, Takao Murakami, and Kamalika Chaudhuri. 2022. Communication-Efficient Triangle Counting under Local Differential Privacy. In *31st USENIX Security Symposium*. 537–554.

[45] Linyu Jiang, Yukun Yan, Zhihong Tian, Zuobin Xiong, and Qilong Han. 2023. Personalized sampling graph collection with local differential privacy for link prediction. *World Wide Web* 26, 5 (2023), 2669–2689.

[46] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.

[47] Muah Kim, Onur Günlü, and Rafael F. Schaefer. 2021. Federated Learning with Local Differential Privacy: Trade-Offs Between Privacy, Utility, and Communication. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2650–2654. https://doi.org/10.1109/ICASSP39728.2021.9413764

[48] Maksim Kitsak, Lazaros K. Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H. Eugene Stanley, and Hernán A. Makse. 2010. Identification of influential spreaders in complex networks. *Nature Physics* 6, 11 (Nov. 2010), 888–893. https://doi.org/10.1038/nphys1746

[49] Tejas Kulkarni. 2019. Answering Range Queries Under Local Differential Privacy. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1832–1834. https://doi.org/10.1145/3299869.3300102

[50] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a Social Network or a News Media?. In *www*. 591–600.

[51] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. (2014).

[52] Xiaoguang Li, Ninghui Li, Wenhai Sun, Neil Zhenqiang Gong, and Hui Li. 2023. Fine-grained poisoning attack to local differential privacy protocols for mean and variance estimation. In *32nd USENIX Security Symposium (USENIX Security*

23). 1739–1756.

[53] Wanyu Lin, Baochun Li, and Cong Wang. 2022. Towards private learning on decentralized graphs with local differential privacy. *IEEE Transactions on Information Forensics and Security* 17 (2022), 2936–2946.

[54] Fang Liu, Dong Wang, and Tian Yan. 2023. Some examples of privacy-preserving sharing of COVID-19 pandemic data with statistical utility evaluation. *BMC Medical Research Methodology* 23, 1 (2023), 120.

[55] Quanquan C. Liu, Jessica Shi, Shangdi Yu, Laxman Dhulipala, and Julian Shun. 2022. Parallel Batch-Dynamic Algorithms for k-Core Decomposition and Related Graph Problems. In *34th ACM Symposium on Parallelism in Algorithms and Architectures*. 191–204.

[56] Shang Liu, Yang Cao, Takao Murakami, Jinfei Liu, and Masatoshi Yoshikawa. 2023. CARGO: Crypto-Assisted Differentially Private Triangle Counting without Trusted Servers. *arXiv preprint arXiv:2312.12938* (2023).

[57] Shang Liu, Yang Cao, Takao Murakami, and Masatoshi Yoshikawa. 2022. A crypto-assisted approach for publishing graph statistics with node local differential privacy. In *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 5765–5774.

[58] Yuhan Liu, Tianhao Wang, Yixuan Liu, Hong Chen, and Cuiping Li. 2024. Edge-Protected Triangle Count Estimation under Relationship Local Differential Privacy. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[59] 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)*. IEEE, 2008–2020.

[60] Zemin Liu, Vincent W. Zheng, Zhou Zhao, Hongxia Yang, Kevin Chen-Chuan Chang, Minghui Wu, and Jing Ying. 2018. Subgraph-Augmented Path Embedding for Semantic User Search on Heterogeneous Social Network. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 1613–1622. https://doi.org/10.1145/3178876.3186073

[61] Pathum Chamikara Mahawaga Arachchige, Dongxi Liu, Seyit Camtepe, Surya Nepal, Marthie Grobler, Peter Bertok, and Ibrahim Khalil. 2022. Local Differential Privacy for Federated Learning. In *European Symposium on Research in Computer Security*. Springer, 195–216.

[62] Naoki Masuda, Michiko Sakaki, Takahiro Ezaki, and Takamitsu Watanabe. 2018. Clustering Coefficients for Correlation Networks. *Frontiers in Neuroinformatics* 12 (2018). https://doi.org/10.3389/fninf.2018.00007

[63] Gang Mei, Jingzhi Tu, Lei Xiao, and Francesco Piccialli. 2021. An efficient graph clustering algorithm by exploiting k-core decomposition and motifs. *Computers & Electrical Engineering* 96 (2021), 107564. https://doi.org/10.1016/j.compeleceng.2021.107564

[64] Tamara T Mueller, Dmitrii Usynin, Johannes C Paetzold, Daniel Rueckert, and Georgios Kaissis. 2022. SoK: Differential privacy on graph-structured data. *arXiv preprint arXiv:2203.09205* (2022).

[65] Takao Murakami and Yuichi Sei. 2023. Automatic Tuning of Privacy Budgets in Input-Discriminative Local Differential Privacy. *IEEE Internet of Things Journal* (2023).

[66] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. 2022. Local and Central Differential Privacy for Robustness and Privacy in Federated Learning. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/auto-draft-204/

[67] Neo4j. 2012. Neo4j - The World's Leading Graph Database. http://neo4j.org/

[68] Dung Nguyen, Mahantesh Halappanavar, Venkatesh Srinivasan, and Anil Vullikanti. 2024. Faster approximate subgraph counts with privacy. *Advances in Neural Information Processing Systems* 36 (2024).

[69] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2007. Smooth Sensitivity and Sampling in Private Data Analysis. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*. 75–84.

[70] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (2005), 814–818.

[71] Arnau Prat-Pérez, David Dominguez-Sal, Josep M Brunat, and Josep-Lluis Larriba-Pey. 2012. Shaping communities out of triangles. In *Proceedings of the ACM international Conference on Information and Knowledge Management*. 1677–1681.

[72] Lei Qin, Yidan Wang, Qiang Sun, Xiaomei Zhang, Ben-Chang Shia, Chengcheng Liu, et al. 2020. Analysis of the covid-19 epidemic transmission network in mainland china: K-core decomposition study. *JMIR public health and surveillance* 6, 4 (2020), e24291.

[73] 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* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 425–438. https://doi.org/10.1145/3133956.3134086

[74] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 29.

[75] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. 2021. Mycelium: Large-scale distributed graph queries with differential privacy. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 327–343.

[76] Higor S Monteiro, Shaojun Luo, Saulo DS Reis, Carles Igual, Antonio S Lima Neto, Matias Travizan, Jose Soares De Andrade Jr, Hernan Makse, et al. 2021. Superspreading k-cores at the center of Covid-19 pandemic persistence. *Bulletin of the American Physical Society* 66 (2021).

[77] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

[78] Mohamed Seif, Dung Nguyen, Anil Vullikanti, and Ravi Tandon. 2022. Differentially private community detection for stochastic block models. *arXiv preprint arXiv:2202.00636* (2022).

[79] Matteo Serafino, Higor S. Monteiro, Shaojun Luo, and Hernán A. Makse. 2020. *Project COVID19 K-core tracker*. https://github.com/makselab/COVID19

[80] Matteo Serafino, Higor S Monteiro, Shaojun Luo, Saulo DS Reis, Carles Igual, Antonio S Lima Neto, Matías Travizano, José S Andrade Jr, and Hernán A Makse. 2022. Digital contact tracing and network theory to stop the spread of COVID-19 using big-data on human mobility geolocalization. *PLOS Computational Biology* 18, 4 (2022), e1009865.

[81] Elaine Shi, T.-H. Hubert Chan, Eleanor Gilbert Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-Preserving Aggregation of Time-Series Data. In *Proceedings of the Network and Distributed System Security Symposium*.

[82] Konstantinos Sotiropoulos and Charalampos E. Tsourakakis. 2021. Triangle-Aware Spectral Sparsifiers and Community Detection. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 1501–1509. https://doi.org/10.1145/3447548.3467260

[83] Sriganesh Srihari and Hon Leong. 2013. A survey of computational methods for protein complex prediction from protein interaction networks. *Journal of bioinformatics and computational biology* 11 (04 2013), 1230002. https://doi.org/10.1142/S021972001230002X

[84] Haipei Sun, Xiaokui Xiao, Issa Khalil, Yin Yang, Zhan Qin, Hui Wang, and Ting Yu. 2019. Analyzing subgraph statistics from extended local views with decentralized differential privacy. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 703–717.

[85] Marco Sánchez-Aguayo, Luis Urquiza-Aguiar, and José Estrada-Jiménez. 2021. Fraud Detection Using the Fraud Triangle Theory and Data Mining Techniques: A Literature Review. *Computers* 10, 10 (2021). https://doi.org/10.3390/computers10100121

[86] Differential Privacy Team. 2017. Learning with Privacy at Scale — machinelearning.apple.com. https://machinelearning.apple.com/research/learning-with-privacy-at-scale. [Accessed 10-04-2024].

[87] Google Differential Privacy Team. [n. d.]. GitHub - google/differential-privacy: Google's differential privacy libraries. — github.com. https://github.com/google/differential-privacy. [Accessed 12-04-2024].

[88] Ekin Tire and M. Emre Gursoy. 2024. Answering Spatial Density Queries Under Local Differential Privacy. *IEEE Internet of Things Journal* 11, 10 (2024), 17419–17436. https://doi.org/10.1109/JIOT.2024.3357570

[89] Tom Tseng, Laxman Dhulipala, and Julian Shun. 2021. Parallel index-based structural graph clustering and its approximation. In *Proceedings of the International Conference on Management of Data*. 1851–1864.

[90] Songlei Wang, Yifeng Zheng, Xiaohua Jia, Qian Wang, and Cong Wang. 2023. MAGO: Maliciously Secure Subgraph Counting on Decentralized Social Graphs. *IEEE Transactions on Information Forensics and Security* (2023).

[91] Tianhao Wang, Bolin Ding, Jingren Zhou, Cheng Hong, Zhicong Huang, Ninghui Li, and Somesh Jha. 2019. Answering Multi-Dimensional Analytical Queries under Local Differential Privacy. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 159–176. https://doi.org/10.1145/3299869.3319891

[92] Yanling Wang, Qian Wang, Lingchen Zhao, and Cong Wang. 2023. Differential privacy in deep learning: Privacy and beyond. *Future Generation Computer Systems* (2023).

[93] Zhuo Wang, Zhixiong Li, Jinxing Tu, and Jianqiang Huang. 2025. DYTC:Dynamic Graph Triangle Counting on GPU. In *Proceedings of the 2024 8th International Conference on Algorithms, Computing and Systems (ICACS '24)*. Association for Computing Machinery, New York, NY, USA, 83–87. https://doi.org/10.1145/3708597.3708610

[94] Stanley L Warner. 1965. Randomized response: A survey technique for eliminating evasive answer bias. *J. Amer. Statist. Assoc.* 60, 309 (1965), 63–69.

[95] Zihang Xiang, Tianhao Wang, and Di Wang. 2023. Preserving Node-level Privacy in Graph Neural Networks. *arXiv preprint arXiv:2311.06888* (2023).

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

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

[98] Wei Zeng, An Zeng, Hao Liu, Ming-Sheng Shang, and Tao Zhou. 2014. Uncovering the information core in recommender systems. *Scientific reports* 4 (August 2014), 6140. https://doi.org/10.1038/srep06140

[99] Da Zhong, Ruotong Yu, Kun Wu, Xiuling Wang, Jun Xu, and Wendy Hui Wang. 2023. Disparate Vulnerability in Link Inference Attacks against Graph Neural Networks. *Proceedings on Privacy Enhancing Technologies* (2023).

[100] Xiaochen Zhu, Vincent YF Tan, and Xiaokui Xiao. 2023. Blink: Link Local Differential Privacy in Graph Neural Networks via Bayesian Estimation. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 2651–2664.