Journal of Privacy and Confidentiality
Vol. 16 (2) 2026
TPDP 2024
Submitted April 2025
Published August 2026

SUBLINEAR SPACE GRAPH ALGORITHMS IN THE CONTINUAL RELEASE MODEL

ALESSANDRO EPASTO, QUANQUAN C. LIU, TAMALIKA MUKHERJEE, AND FELIX ZHOU

Google Research, New York, USA
e-mail address: aepasto@google.com

Yale University, New Haven, USA
e-mail address: quanquan.liu@yale.edu

Max Planck Institute for Security and Privacy, Bochum, Germany
e-mail address: tamalika.mukherjee@mpi-sp.org

Yale University, New Haven, USA
e-mail address: felix.zhou@yale.edu

Abstract. The graph continual release model of differential privacy seeks to produce differentially private solutions to graph problems under a stream of edge updates where new private solutions are released after each update. Previously known *edge differentially private* algorithms for most graph problems including densest subgraph and matchings in the continual release setting only output real-valued estimates (not vertex subset solutions) and do not use sublinear space. In this paper, we leverage *sparsification* to address the above shortcomings for edge-insertion streams. Our edge differentially private algorithms use sublinear space with respect to the number of edges in the graph. In addition, for the densest subgraph problem, we output edge differentially private vertex subset solutions; no previous graph algorithms in the continual release model output such subsets.

We make novel use of sparsification techniques from the non-private streaming and static graph algorithms literature to achieve new results in the sublinear space continual release setting. This includes algorithms for densest subgraph, maximum matching, and the first continual release k-core decomposition algorithm. To complement our insertion-only algorithms, we conclude with polynomial additive error lower bounds for edge-privacy in the fully dynamic setting, where only logarithmic lower bounds were previously known.

1. Introduction

Today's data is not only massive in size but also rapidly growing. As the world becomes increasingly digitized, the sensitive associations between individuals are becoming a key focus for data analysis. Such a focus also comes with privacy risks and the question of how to safeguard privacy. Differential privacy (DP) (Dwork et al., 2006) is the gold standard for privacy protection, with a rich body of work focusing on *static* graphs (Arora and

Key words and phrases: Continual Release, Streaming Algorithms, Online Algorithms, k-Core Decomposition, Densest Subgraph, Matchings.

www.journalprivacyconfidentiality.org
DOI:10.29012/jpc.997

Upadhyay, 2019; Blocki et al., 2022; Dinitz et al., 2025a; Dhulipala et al., 2022; Eden et al., 2023; Kasiviswanathan et al., 2013; Kalemaj et al., 2023; Liu et al., 2024; Mueller et al., 2024; Nissim et al., 2007; Raskhodnikova and Smith, 2016a,b; Upadhyay, 2013). However, today's graphs are not static; they are rapidly expanding, with new connections between individuals forming every minute. Streaming graph algorithms, a well-studied area in the non-private setting (Assadi et al., 2022; Ahn and Guha, 2011; Esfandiari et al., 2016, 2018; Feigenbaum et al., 2005; Fischer et al., 2022; Huang and Peng, 2019; Henzinger et al., 1998; Muthukrishnan et al., 2005; McGregor, 2014; McGregor et al., 2015; McGregor and Vorotnikova, 2018), are designed to handle such massively evolving datasets. In this model, updates to the dataset are received in a stream. Online streaming algorithms continuously release accurate graph statistics after each update (Chen et al., 2023b; Halldórsson et al., 2016; Cormode et al., 2018; Ghosh and Stoeckl, 2024), but these graph statistics may reveal private information.

Our paper focuses on studying graph problems in the well-known *continual release* model (Dwork et al., 2010a; Chan et al., 2011) of differential privacy that promises a strong guarantee for online streaming outputs: the entire vector of *all outputs* satisfies differential privacy. The notion of graph privacy we consider is edge-differential privacy, where we require the output distributions of the algorithm on edge-neighboring stream inputs to be "close", where edge-neighboring streams differ in one edge update. Despite the large body of differential privacy works on static graph algorithms, there are few works on graph continual release (Fichtenberger et al., 2021; Jain et al., 2024; Song et al., 2018) and related graph models (Upadhyay et al., 2021). While there has been substantial progress in graph continual release algorithms in recent years, there are some drawbacks to existing algorithms, including: 1) not releasing private (vertex subset) solutions in addition to their real-valued function outputs and 2) requiring exact algorithms for computing graph statistics, thus necessitating linear space usage in the number of *edges*.

In this paper, we take the first steps to address these shortcomings for insertion-only streams in the continual release model via the following contributions:

- We give the first DP k-core decomposition algorithm in the continual release model.
- We formulate the first continual release graph algorithms that return edge-DP vertex subset solutions for densest subgraph (in addition to the value of the solutions).
- We give the first *sublinear* space DP algorithms for general graphs to solve densest subgraph, k-core decomposition, and matching problems.

The approximation guarantees of our algorithms nearly match the guarantees for their private static or non-private streaming counterparts. Moreover, as is standard in streaming, all of our results hold for a single pass over the stream.

Resolving these issues closes the gap between continual release graph algorithms and their non-private streaming counterparts for many applications. While the private streaming literature has explored sublinear space outside the graph algorithms setting (Chan et al., 2012; Epasto et al., 2023; Upadhyay, 2019; Blocki et al., 2023), it remains unaddressed in general graphs for most problems in continual release. Furthermore, while streaming algorithms typically operate on datasets too large for memory, practical applications often require information about each vertex, not just aggregate values. For example, advertisers may want the members of the current densest community, not merely its density. Hence, achieving both goals–sublinear space combined with useful vertex information–is essential to the development of private graph algorithms.

1.1. **Sparsification.** The main algorithmic technique we use in this paper is *sparsification*, traditionally used in the streaming, static, and dynamic graph algorithms literature. Sparsification aims to simplify a large graph by strategically selecting a smaller set of its edges. This smaller "sketch" of the graph still captures the essential information needed for a specific problem. Such sparsifiers exist for a large number of problems in the non-private literature. In general, graph sparsifiers are *problem-specific*, where each sparsifier provides guarantees for only one or two closely related problems. We design novel methods for creating differentially private problem-specific graph sparsifiers in continual release, which we describe in more detail in the technical overview (Section 2).

Using sparsification techniques, we design a number of new continual release algorithms that achieve the same utility guarantees as previous algorithms for the problem in the static DP and non-private streaming settings up to logarithmic factors. Our sparsification techniques directly lead to space savings in the edge-privacy setting for a variety of problems including densest subgraph, k-core decomposition, and maximum matching. We return vertex subsets for the densest subgraph problem, approximate k-core numbers for every vertex in core decomposition, and estimates for the size of the maximum matching, all using sublinear space in the *number of edges*. For maximum matching, we even use space that is sublinear in the number of vertices.[1]

Finally, to complement our results on insertion-only streams, we demonstrate the difficulty of obtaining algorithms in fully dynamic streams via improved polynomial error lower bounds for maximum cardinality matching, counting connected components, and triangle counting based on reductions from inner product queries. Similar to the recent continual release lower bounds for counting distinct elements due to Jain et al. (2023a), our lower bounds support the observation from previous works that the fully dynamic setting results in significantly more error (Song et al., 2018; Fichtenberger et al., 2021) than the insertion-only setting within the continual release model.

1.2. **Our Contributions.** Our paper gives the following set of results, stated informally here and given in more precise terms in their respective sections. We group our results by graph problems for n-vertex m-edge graph streams over T timesteps and our multiplicative guarantees are given in terms of a constant factor $\eta > 0$.

For our algorithmic results, we consider insertion-only streams where (possibly empty) edge insertions are given in the stream. For simplicity, we assume that each non-empty edge is inserted at most once. Without requiring privacy, this closely resembles the well-studied online streaming setting (see related works in Section 3). Our algorithms also incur logarithmic additive noise, which is necessary, as demonstrated by lower bounds that hold even in the static model (see Table 1).

Our lower bounds are given for fully dynamic streams where edges can be inserted and deleted multiple times, also known as the *turnstile* model. In this setting, we show that polynomial additive error is necessary, demonstrating a fundamental difficulty in maintaining small errors in the presence of both insertions and deletions.

All of our results in the continual release model are given for edge-neighboring streams where the two streams differ by one edge update; results in the static DP setting are given

[1]While all non-private streaming algorithms use space that is sublinear in the number of total edges, very few use space sublinear in the number of vertices, although results sublinear in the number of vertices are most desirable.

for edge-neighbors (graphs that differ by one edge). Throughout the paper, we will use the phrase *ε-edge DP* to refer to these settings. Our algorithms produce (β, ζ)-bicriteria approximations where β is the multiplicative factor in the approximation and ζ is the additive error. Our approximation bounds hold, with high probability, for the released solution after *every* update in the stream. Throughout the paper, we assume $n \leq T = O(\text{poly}(n))$. This is reasonable as $O(n^2)$ (non-empty) updates are sufficient to have a complete graph.[2] We compare our results with static edge-DP lower bounds as such lower bounds also hold in the insertion-only continual release setting.[3] A more in-depth discussion of related works can be found in Section 3. Our entire set of results can be found in Table 1, including baseline algorithms and lower bounds in the static edge-DP and non-private streaming settings.

k-Core Decomposition (Section 5). We show the first ε-edge DP algorithm for the k-core decomposition problem in the continual release model. Our algorithm returns approximate k-core numbers at every timestep while using $o(m)$ space. The additive errors of our algorithms are near-optimal as they match the recent additive error lower bounds up to logarithmic factors (see Table 1). We note that these lower bounds hold even when allowing for constant multiplicative error.

Theorem 1.1 (See Theorem 5.1)**.** *We obtain a $\left(2+\eta, O\left(\frac{\log^3(n)}{\eta^2\varepsilon}\right)\right)$-approximate ε-edge differentially private k-core decomposition algorithm that outputs core number estimates in the insertion-only continual release model using $\tilde{O}\left(\frac{n}{\eta^4\varepsilon}\right)$ space.*

The essence of our k-core algorithm is a novel *sparse* level data structure that may be of independent interest beyond our work. To the best of our knowledge, such a data structure did not previously exist in the privacy or graph algorithms literature. The streaming k-core algorithm of Esfandiari et al. (2018) computes a randomized sketch of the input graph and runs the classic peeling algorithm on the sketch rather than maintain a level data structure.

Densest Subgraph (Section 6). We obtain the first ε-edge DP densest subgraph algorithms that return subsets of vertices with near-optimal density in the continual release model. Our algorithms also use $o(m)$ space. We emphasize that the approximation guarantees nearly match, up to logarithmic terms, the guarantees of their private static or non-private streaming counterparts. In turn, these nearly match the information-theoretic lower bound up to logarithmic factors. Again, we emphasize that these lower bounds hold even when allowing for constant multiplicative error.

Theorem 1.2 (See Theorem 6.1)**.** *We obtain a $\left(2+\eta, O\left(\frac{\log^2(n)}{\eta^2\varepsilon}\right)\right)$-approximate ε-edge differentially private densest subgraph algorithm that outputs a subset of vertices in the insertion-only continual release model using $\tilde{O}\left(\frac{n}{\eta^2\varepsilon}\right)$ space.*

At the cost of slightly more additive error and space usage, we can improve the multiplicative error.

[2] Our results can be modified to handle streams of length $T = \omega(\text{poly}(n))$.

[3] Otherwise, if we obtain better error in the continual release setting, we can solve the static problem with better error by inserting all of the edges of the static graph and taking the solution released at the end of the stream.

TABLE 1. Our results are highlighted in green. Non-private bounds are shown in gray. All results are for edge-privacy. Bicriteria approximations (β, ζ) are given, where β is the multiplicative factor and ζ is the additive error. $\tilde{\alpha}$ is a public upper bound on the maximum arboricity of the stream; our approximation guarantees for maximum matching hold when $\tilde{\alpha}$ upper bounds the private arboricity and our privacy guarantees *always* hold. α in the non-private streaming bound is the maximum arboricity in the stream.

Problem	Space	Approximation	Reference	Model	Solution Type
k-Core Decomp.	$\tilde{O}\left(\frac{n}{\eta^4\varepsilon}\right)$	$\left(2+\eta, O\left(\frac{\log^3(n)}{\eta^2\varepsilon}\right)\right)$	Theorem 5.1	Insertion Only	Upper Bound
	$\Theta(m+n)$	$\left(1, O\left(\frac{\log(n)}{\varepsilon}\right)\right)$	Dhulipala et al. (2023)	Static	Upper Bound
		$\left(\beta, \Omega\left(\frac{\log n}{\varepsilon\beta}\right)\right)$	Henzinger et al. (2024a)	Static	Lower Bound
	$\tilde{O}\left(\frac{n}{\eta^2}\right)$	$(1+\eta, 0)$	Esfandiari et al. (2018)	Non-Private Streaming	Upper Bound
Densest Subgraph	$\tilde{O}\left(\frac{n}{\eta^2\varepsilon}\right)$	$\left(2+\eta, O\left(\frac{\log^2(n)}{\eta\varepsilon}\right)\right)$	Theorem 6.1	Insertion Only	Upper Bound Vertex Subset
	$\tilde{O}\left(\frac{n}{\eta^5\varepsilon}\right)$	$\left(1+\eta, O\left(\frac{\log^5(n)}{\eta^4\varepsilon}\right)\right)$	Theorem 6.2	Insertion Only	Upper Bound Vertex Subset
	$\Theta(m+n)$	$\left(1+\eta, O\left(\frac{\log^4(n)}{\eta^3\varepsilon}\right)\right)$	Dhulipala et al. (2022)	Static	Upper Bound Vertex Subset
	$\Theta(m+n)$	$\left(2, O\left(\frac{\log(n)}{\varepsilon}\right)\right)$	Dhulipala et al. (2023)	Static	Upper Bound Vertex Subset
	$\Theta(m+n)$	$\left(2+\eta, O\left(\frac{\log^2(n)}{\eta\varepsilon}\right)\right)$	Dinitz et al. (2025a)	Static	Upper Bound Vertex Subset
		$\left(\beta, \Omega\left(\frac{1}{\beta}\sqrt{\frac{\log(n)}{\varepsilon}}\right)\right)$	Farhadi et al. (2022)	Static	Lower Bound
	$\Theta(m)$	$\left(1+\eta, O\left(\frac{\log^2(n)}{\eta\varepsilon}\right)\right)$	Fichtenberger et al. (2021)	Insertion Only	Upper Bound Density Only
	$\tilde{O}\left(\frac{n}{\eta^2}\right)$	$(1+\eta, 0)$	McGregor et al. (2015) Esfandiari et al. (2016)	Non-Private Streaming	Upper Bound
Maximum Matching Size	$O\left(\frac{\log^2(n)\log(\tilde{\alpha})}{\eta^2\varepsilon}\right)$	$\left((1+\eta)(2+\tilde{\alpha}), O\left(\frac{\log^2(n)}{\eta\varepsilon}\right)\right)$	Theorem 7.1	Insertion Only	Upper Bound Arboricity
	$\Theta(m)$	$\left(1+\eta, O\left(\frac{\log^2(n)}{\eta\varepsilon}\right)\right)$	Fichtenberger et al. (2021)	Insertion Only	Upper Bound
		$(1, \Omega(\log(T)))$	Fichtenberger et al. (2021)	Insertion Only	Lower Bound
		$\left(1, \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right)\right)$	Theorem 8.1	Fully Dynamic	Lower Bound
		$(1, \Omega(\log(T)))$	Fichtenberger et al. (2021)	Fully Dynamic	Lower Bound
	$O\left(\frac{\log(n)\log(\alpha)}{\eta^2}\right)$	$((1+\eta)(2+\alpha), 0)$	McGregor and Vorotnikova (2018)	Non-Private Streaming	Upper Bound Arboricity
Triangle Count		$\left(1, \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right)\right)$	Theorem 8.2	Fully Dynamic	Lower Bound
		$(1, \Omega(\log(T)))$	(Fichtenberger et al., 2021)	Fully Dynamic	Lower Bound
Connected Components Counting		$\left(1, \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right)\right)$	Theorem 8.3	Fully Dynamic	Lower Bound

Theorem 1.3 (See Theorem 6.2)**.** *We obtain a* $\left(1+\eta, O\left(\frac{\log^5(n)}{\eta^4\varepsilon}\right)\right)$*-approximate* ε*-edge differentially private densest subgraph algorithm that outputs a subset of vertices in the insertion-only continual release model using* $\tilde{O}\left(\frac{n}{\eta^5\varepsilon}\right)$ *space.*

Maximum Matching (Section 7). We give the following results for estimating the size of a maximum matching in bounded arboricity graphs. Arboricity, a measure of local sparsity in the input graph, is formally defined as the minimum number of forests into which a graph's edges can be partitioned. In real-world graphs, the arboricity of a graph is typically $\text{poly}(\log(n))$. In this setting, $\tilde{\alpha} > 0$ is a given public bound on the maximum arboricity α of the graph given by the input stream.[4] We also briefly sketch how to remove this assumption in Appendix I.2 at the cost of more space and a worse approximation guarantee. The additive error of our algorithms matches the information-theoretic lower bound in bounded arboricity graphs up to logarithmic factors.

Our matching algorithm falls under the *truly sublinear model*, where space usage is sublinear in the *number of nodes* in the graph. The truly sublinear model is the most difficult streaming model to obtain algorithms in, and very few graph algorithms, even in the non-private setting, use truly sublinear space. We give an edge-DP algorithm for estimating the size of a maximum matching in bounded arboricity graphs. Our approximation guarantees hold when our public bound $\tilde{\alpha}$ upper bounds the arboricity of the graph, i.e. $\tilde{\alpha} \geq \alpha$, while our privacy guarantees always hold.

Theorem 1.4 (See Theorem 7.1)**.** *Given a public bound $\tilde{\alpha}$, we obtain an ε-edge differentially private approximate maximum cardinality matching algorithm that outputs an approximate matching size using $O\left(\frac{\log^2(n)\log(\tilde{\alpha})}{\eta^2\varepsilon}\right)$ space in the insertion-only continual release model. If $\tilde{\alpha} \geq \alpha$, where α is the (private) maximum arboricity of the input graph, then our algorithm returns a $\left((1+\eta)(2+\tilde{\alpha}), O\left(\frac{\log^2(n)}{\eta\varepsilon}\right)\right)$-approximation of the size of the maximum matching.*

Fully Dynamic Lower Bounds (Section 8). Last but not least, we give improved lower bounds in the fully-dynamic setting in Section 8 for estimating the size of a maximum matching and even simpler problems like connected component counting and triangle counting. Our reductions follow a natural framework that generalizes to further problems (Section 8.1).

Note that previous lower bounds all show a $\Omega(\text{poly}\log(T))$ lower bound in the additive error while we strengthen these lower bounds to $\Omega(\text{poly}(T))$, yielding an exponential improvement. Comparing the upper bounds with our new lower bounds for the same problems yields polynomial separations in the error between the insertion-only and fully dynamic settings.

See Table 1 for a summary of our results and previous works.

2. Technical Overview

In this section, we provide a detailed overview of our techniques. We begin in Section 2.1 by illustrating the various algorithmic challenges with the graph continual release model. Section 2.2 then describes how we address these challenges.

2.1. Challenges in Continual Release.

[4]Such public bounds are commonly assumed in non-private streaming literature (Cormode et al., 2017; McGregor and Vorotnikova, 2018).

Returning Multiple Solutions. The best one-pass streaming algorithms in the non-private setting for the problems we study (Esfandiari et al., 2018; McGregor and Vorotnikova, 2018; McGregor et al., 2015; Esfandiari et al., 2016) return the solution only once, at the end of the stream. In the continual release setting, we must return a solution *after every update*. This is a fundamental challenge, especially when considering privacy. Indeed, with a single solution release, each edge update is intuitively used only once to produce the solution. In contrast, releasing solutions multiple times means earlier updates are potentially used many times to produce solutions for each release. Such overuse could result in greater loss of privacy. Thus, for our continual release algorithms, we use novel techniques to simultaneously ensure we achieve edge-privacy for the entire vector of releases as well as ensure our utility bounds for each release.

Previous value-only continual release algorithms ensure privacy over multiple releases via the sparse vector technique (SVT). These algorithms release the outputs of one or more instances of SVT over the stream and the estimates are a deterministic function of the SVT outputs. Thus, the proof of privacy directly follows from the privacy guarantees of SVT. The outputs of our algorithms cannot be directly derived from standard SVT as we need to output information about individual vertices (as part of vertex subsets or individual outputs for each vertex for k-core decomposition). Moreover, our outputs depend on our sampling procedures, which in turn depend on private properties of the stream. These subtleties prevent the use of straightforward analyses of privacy using composition. Instead, we must directly prove the privacy of our algorithms using the definition of edge-privacy and analyze the output distributions of our algorithms over the entire stream. This is accomplished via careful conditioning arguments.

Ensuring Sublinear Space. Obtaining sublinear space continual release algorithms presents a new set of challenges. In particular, we must ensure that our sampling techniques are *stable*.[5] Stable sampling algorithms use sampling techniques that do not cause our samples from neighboring streams to differ by more than one edge update. For randomized sampling algorithms, this means that the distribution of sampled edges from edge-neighboring streams should be similar. Consider the following naive sampling scheme that is not stable. Suppose we want to sample each edge insertion in an insertion-only stream with probability proportional to the number of edges we have seen so far. Thus, the i-th inserted edge will be sampled with probability $1/i$. Suppose we have two neighboring streams S and S' where the edge that differs is the first update; thus, the first update in S' is an edge insertion, while the first update in S is $\perp$ (an empty update). Then, the j-th update in S, if it is an edge insertion, has a higher probability of being sampled than the j-th update in S'. Such a discrepancy produces significantly different output distributions when sampling from neighboring streams, which may lead to sampled graphs that significantly differ.

Hence, a major component of our algorithms and proofs is to formulate and prove stable sampling algorithms with similar output distributions on edge-neighboring streams. Common sampling approaches in the non-private streaming and sparsification literature do not immediately translate to stable algorithms in the continual release setting.

We note that there are cases where it is known that subsampling amplifies privacy guarantees (Balle et al., 2018). However, since the subsampling probabilities in our algorithms change throughout, it is difficult to use amplification by subsampling.

[5]This term was also used in node-private continual release literature (Jain et al., 2024), i.e. edge-to-edge and node-to-edge stability.

k-Core Decomposition. Consider the k-core decomposition problem, which is defined in terms of the *cores* of a graph. A k-core is a maximal induced subgraph where every vertex in the induced subgraph has degree at least k. Then, a vertex v's core number is the largest value of k for which v is part of the k-core. In edge-neighboring graphs, the removal of one edge can change the core numbers of *all* vertices. Hence, we cannot use a function that computes the exact core number of each node in the continual release setting, as this can incur $\Omega(n)$ error through composition over the vertices. In the continual release setting, the insertion of a single edge can alter the core number of one or more vertices; hence, we also cannot run a static differentially private k-core decomposition algorithm each time a core number changes since we can potentially see $\Omega(n)$ core number changes incurred over all vertices. Calling the static DP algorithm $\Omega(n)$ times would result in $\Omega(n)$ error through composition over the calls. To overcome these challenges, we develop a novel k-core decomposition algorithm using several new techniques (Section 5).

Releasing Vertex Subsets. Now, consider the setting where we return vertex subset solutions. Unlike returning real-valued solutions, even a few updates that do not significantly alter the solution's value (e.g. the maximum density in the densest subgraph problem) can still cause the entire set of vertices to change. This can happen even when the global sensitivity is small. Such behavior differs from real-valued solutions, which are less susceptible to large changes due to small updates. To give a concrete example, consider the densest subgraph problem where the goal is to obtain a subset of vertices with maximum induced density. Suppose we are given a number of edge insertions resulting in two disjoint almost k-cliques, A and B, that is, two k-cliques, each with 2 edges removed. A sequence of insertions—one into A followed by two into B—could lead to large changes in the set of vertices in the densest subgraph. After the first insertion, the densest subgraph consists of vertices in A; then, after the third insertion, the densest subgraph consists of vertices in B. This sequence of only 3 insertions causes a complete change in the optimal vertex subset. It is not hard to extend this example and show that we can force any $O(1)$-approximation algorithm to completely change its output with $O(1)$ insertions. Because of this challenge and the privacy requirement, we allow for $(1+\eta)$-multiplicative approximations along with $\text{poly}(\log n)/\varepsilon$ additive error on the density of the subgraph induced by the vertex subsets we release.

To release a private densest subgraph, we can use a static ε-edge DP densest subgraph algorithm that releases the vertex subset. However, as mentioned above, we cannot afford to release a new private vertex subset each time we receive an update, even if the true vertex subset changes, since we would incur $\Omega(n)$ error from composition. Thus, our algorithms in Section 6 determine appropriate timesteps for releasing new vertex subsets.

2.2. **Detailed Description of Our Techniques.** All of our algorithms use $\tilde{O}(n)$ space, i.e., near-linear in the number of vertices in the stream, and some are sublinear in the number of vertices. We present ε-differentially private algorithms in edge-neighboring streams (Definition 4.2), where the streams differ by one edge update.

We now describe how we solve each of the aforementioned challenges in Section 2.1 and obtain algorithms for a variety of problems in the next sections. Each of these problems uses a unique sampling scheme, as sparsifiers in graph literature are often problem-dependent. We prove the privacy of every algorithm as well as the utility of each of our algorithms within their individual sections. We summarize on a high level our technical contributions for each of the following problems.

k-Core Decomposition (Section 5). Given a stream of updates S, the k-core decomposition problem asks for a number for each vertex after every update indicating the maximum value k for which the vertex is contained in a k-core. A k-core is defined as a maximal set of vertices $U \subseteq V$ such that the degree of every vertex in the subgraph induced by U is at least k. We return approximate core numbers for every vertex at every timestep t that are $\left(2+\eta, O\left(\frac{\log^3(n)}{\eta^2\varepsilon}\right)\right)$-approximations of their true core numbers (see Theorem 1.1) while using $\tilde{O}\left(\frac{n}{\eta^4\varepsilon}\right)$ total space.

We formulate a novel differentially private continual release algorithm for k-core decomposition that is loosely inspired by the level data structure of the static $\left(2+\eta, O\left(\frac{\log^3(n)}{\eta^2\varepsilon}\right)\right)$-approximation algorithm of Dhulipala et al. (2022). As described in Section 2.1, we cannot use the static private algorithm in a black-box manner since we need to return the value of *every* vertex at each timestep. Every vertex may change its core number many times throughout the duration of the stream. Thus, using any static k-core decomposition algorithm in a black-box manner incurs a privacy loss of a factor of n via composition.

In the static setting, the level data structure algorithm for k-core decomposition partitions the vertices of the input graph across the levels based on the induced degree of each vertex among the vertices in the same or higher levels. Then, the levels roughly partition the vertices into cores of the graph with higher levels containing larger valued cores and lower levels containing smaller valued cores. In the static setting, vertices move up levels using all of the edges in the graph for each move. However, in the continual release setting, edges are inserted into the graph progressively and vertices move up the levels as new edges are inserted. In the static setting, each vertex releases its level at most $\text{poly}(\log n)$ times, whereas in the continual release setting, each vertex releases its approximate core number $\Omega(T) = \text{poly}(n)$ times. While composition over $\text{poly}(\log n)$ releases leads to $\text{poly}(\log n)$ additive error, composition over $\Omega(n)$ releases in the continual release model would lead to $\Omega(n)$ error.

Due to the above challenge, we need to be able to have vertices release their approximate core numbers $\Omega(T)$ times without losing a factor of T in the additive error due to composition. This problem is not present in the static setting. In our novel continual release algorithm, we use a variant of the *sparse vector technique* adapted to the level data structure. In particular, we use the idea of the *multidimensional sparse vector technique* due to Dhulipala et al. (2023) and adapt it to the continual release setting with a new formulation of the algorithm and a new proof allowing for adaptive queries based on new (subsampled) edge insertions to the graph. We call this the *adaptive multidimensional sparse vector technique*.

In the static setting, we can use the multidimensional sparse vector technique to determine when a vertex stops moving up levels. The level in which a vertex then resides directly corresponds with an approximation of the core number of the vertex. Thus, once a vertex stops moving up levels in the static setting, it will immediately output a new approximation. Hence, each vertex fails the SVT check at most once. However, in the continual release setting, our usage of the adaptive multidimensional sparse vector technique requires that each vertex must output a value at every timestep. This means that the adaptive multidimensional sparse vector technique must answer adaptive queries resulting from edge insertions that occur after previous levels are released. Moreover, the mechanism needs to allow for $\text{poly}(\log n)$ above-threshold checks for each vertex, indicating whether the vertex moved up one of the $\text{poly}(\log n)$ levels.

In addition to solving the above challenge, we further present a sublinear space algorithm for k-core decomposition, which uses $\tilde{O}\left(\frac{n}{\eta^4\varepsilon}\right)$ space. The main workhorse is a *sparse* level data structure. No sparsified versions of the level data structure existed in either the non-private graph literature or the private graph literature. While many non-private sparsification algorithms in the streaming model sample edges uniformly at random, such sampling techniques are *insufficient* for k-core decomposition. When all edges are sampled with the same fixed probability, vertices in smaller cores are expected to have no adjacent edges included in the sample. If no adjacent edges are included in the sample for vertex v, then we cannot approximate v's core number. Hence, we employ a more sophisticated sampling algorithm; specifically, we sample each inserted edge in the stream with probability inversely proportional to the level of its lower-level endpoint. This means that edges on higher levels are sampled with a smaller probability than edges on lower levels. In particular, such a sampling scheme ensures that edges in larger valued cores–which inherently contain more edges–are sampled using smaller probabilities than edges in smaller valued cores. Hence, such a sampling scheme simultaneously ensures sublinear space while maintaining a large enough sample of adjacent edges to each vertex to satisfy our approximation bounds on the core numbers of *all* vertices.

Finally, we prove our approximation bounds using a careful Chernoff bound argument on our sampled sets of edges that takes into account the noise resulting from our usage of DP mechanisms, including the adaptive multidimensional sparse vector technique. Our approximation bound uses an original analysis that shows that vertices on a given level in our sparsified structure will be on approximately the same level, with high probability, as in the non-sparsified structure. Thus, the approximation bounds that we obtain from our non-sparsified structure also hold for our sparsified structure.

Densest Subgraph (DSG) (Section 6). Given an edge-neighboring stream and timestamp t, the densest subgraph in G_t is a subset of vertices V_{OPT} which maximizes the density of the induced subgraph, $V_{\text{OPT}} = \arg\max_{V' \subseteq V} \left(\frac{|E_t(V')|}{|V'|}\right)$. We return a subset of vertices V_{approx} whose induced subgraph gives a $\left(1+\eta, O\left(\frac{\log^5(n)}{\eta^4\varepsilon}\right)\right)$-approximation of the density of the densest subgraph (see Table 1) in $\tilde{O}\left(\frac{n}{\eta^5\varepsilon}\right)$ space. Our algorithm calls the static ε-DP DSG algorithm of Dhulipala et al. (2022) in a black-box manner. Using a different static ε-edge DP DSG algorithm (Dhulipala et al., 2023), we can obtain a $\left(2+\eta, O\left(\frac{\log^2(n)}{\eta\varepsilon}\right)\right)$-approximation while saving a polylogarithmic factor in space.

Our continual release densest subgraph algorithm takes inspiration from the non-private $(1+\eta)$-approximate sparsification algorithms of Esfandiari et al. (2016); McGregor et al. (2015), although we must solve several crucial challenges (Section 2.1) to ensure privacy. Indeed, to ensure accurate approximation guarantees, we present a novel concentration bound proof based on an intricate Chernoff bound argument that accounts for errors introduced by our various private subroutines. Previous works in the continual release model only release the *value* of the densest subgraph and not a *vertex subset* (Fichtenberger et al., 2021; Jain et al., 2024). Since the density of the densest subgraph has sensitivity 1, previous works use the sparse vector technique (Dwork et al., 2009; Roth and Roughgarden, 2010; Hardt and Rothblum, 2010; Lyu et al., 2017) and private prefix sums to release these values after adding appropriate Laplace noise. In our work, we release a private set of vertices at timestep t

whose induced subgraph is an approximation of the densest subgraph in G_t.[6] We use the sparse vector technique to determine when to release a new private set of vertices, and we use one of the many existing differentially private densest subgraph algorithms (Farhadi et al., 2022; Dhulipala et al., 2023, 2022; Nguyen and Vullikanti, 2021; Dinitz et al., 2025a) to return a vertex subset in the sparsified graph. However, there are several challenges in simply returning the subgraph obtained from these static algorithms. First, compared to the non-private setting, we cannot obtain an exact densest subgraph in the sparsified graph. Hence, we must ensure that neither the additive nor multiplicative error blows up when the subgraph obtained in the sparsified graph is scaled up to the original graph. To solve this issue, we ensure that the first subgraph we release has size (approximately) at least the additive error returned by the private static algorithms.

There is another important challenge when making our algorithm use sublinear space. Previous non-private algorithms (McGregor et al., 2015; Esfandiari et al., 2016) sample using a *fixed* probability because 1) they assume knowledge of the total number of updates in the stream and 2) only produce the approximate densest subgraph once, at the end of the stream. However, in our setting, we must release an approximate densest subgraph after each update. Furthermore, we do not know the total number of edges in the stream. Thus, we adaptively adjust our probability of sampling as we see more edge insertions. This must be carefully implemented since a naive adjustment procedure may produce an unstable algorithm (as described in Section 2.1).

Towards solving all of these challenges, our sparsification algorithm works as follows. We sparsify the stream by sampling each edge in the stream with probability p_t. Earlier on in the stream, our probability of sampling should be set high enough in order to sample enough edges to ensure our concentration bounds. However, as we see more edges, we reduce the probability of sampling to ensure our sublinear space bounds. In edge-neighboring streams, such reductions in probability may not occur at the same time, thus leading to sparsified graphs that are *not* stable. Hence, we use the sparse vector technique to determine when to reduce our probability of sampling an edge. We ensure privacy based on a careful conditioning on the sampling probabilities. Intuitively, the sparse vector technique ensures that we decrease our sampling probability at the same timestamps in edge-neighboring streams. Conditioning on the timestamps when the sampling probabilities are reduced, we can show that our sampled streams are stable.

Unlike the static setting where the privacy analysis of algorithms that rely on an SVT instance can apply simple composition, we need to derive a probabilistic proof "from scratch" since we must output a solution after every update but only lose privacy on timestamps where the SVT answers "above". Furthermore, our analysis must take into account our usage of two SVTs, one for determining the probabilities of sampling and the other for determining when we release a new vertex subset. The second use of SVT depends on the outputs of the first; that is, determining when to release a new vertex subset depends on our sampling probabilities. Such SVTs lead to errors that affect our approximation guarantees and our careful Chernoff bound argument must account for these errors while maintaining our approximation guarantees.

Maximum Cardinality Matching Size (Section 7). Given a stream of updates S, the maximum cardinality matching problem asks for an integer at each timestep t equal to the size of the maximum matching in G_t. We give an algorithm that uses sublinear space in the

[6] the induced subgraph consists of all updates that occurred on or before timestep t

number of vertices in the graph with edge-DP guarantees in the continual release model. We provide approximation guarantees using a certain property of the input graph stream: the maximum arboricity of the stream. The arboricity of a graph is a measure of local sparsity. Formally, a graph's arboricity, α, is the minimum number of forests into which its edges can be partitioned. Our algorithm uses a public bound $\tilde{\alpha}$. If this public bound upper bounds the private maximum arboricity of the input stream, our algorithm gives approximation guarantees in terms of $\tilde{\alpha}$. We emphasize that our algorithm is always private for all inputs. Specifically, if $\tilde{\alpha}$ upper bounds the private maximum arboricity of the input stream, then we obtain a $\left((1+\eta)(2+\tilde{\alpha}), O\left(\frac{\log^2 n}{\eta\varepsilon}\right)\right)$-approximation of the matching size.

For this algorithm, we take inspiration from the sublinear space non-private $((1+\eta)(2+\alpha))$-approximate maximum matching algorithm of McGregor and Vorotnikova (2018), which also requires an upper bound α on the maximum arboricity of the stream. Their approximation guarantees are given in terms of this upper bound. To the best of our knowledge, the work of McGregor and Vorotnikova (2018) is the only truly sublinear matching algorithm beyond bipartite graphs that terminates after a single pass, as required by continual release.

The key observation that McGregor and Vorotnikova (2018) make is that maintaining $\Omega(\log n/\eta^2)$ edge samples, sampled uniformly at random in the stream, and counting the number of edges in the sample which are adjacent to at most $\alpha+1$ edges occurring later in the stream is sufficient for providing a good estimate of the maximum matching size in terms of the arboricity upper bound. Edges that are adjacent to more than $\alpha+1$ edges that occur later in the stream are discarded from the sample. Unfortunately, in edge-neighboring streams, we cannot add noise to the counts of every edge in the sample since the sensitivity of such counts is $\Omega(\tilde{\alpha})$ (our public bound). Instead, we show through a precise charging argument that the sensitivity of the size of the sample is bounded by 2. Since the sensitivity of the size of the sample is bounded by 2, we can use the sparse vector technique (SVT) to determine when to release a new estimate when the estimate changes by a large amount. Using SVT results in noisy sample sizes which require a new analysis for the utility bounds.

Previous approximation bound analyses (McGregor and Vorotnikova, 2018) relied on the intuition that edge samples approximately fall into buckets determined by the sampling probability. However, with the use of SVT, it is no longer clear which buckets the sampled edges fall into as SVT introduces an additive $O(\log(n)/\varepsilon)$ noise. Thus, to prove our approximation bounds, we first argue that SVT only causes an additive error of $\tilde{O}\left(\frac{\log^2(n)}{\eta\varepsilon}\right)$ on the estimated number of sampled edges; hence, initializing a bucket with sufficiently large value ensures that the bucket for sampled edges with noise does not deviate by too much from the bucket the sampled edges would fall into without noise. The proof of this property requires a detailed casework analysis. We can further remove the assumption of $\tilde{\alpha}$ using additional space via the parallel guessing trick in Appendix I.2.

Fully Dynamic Lower Bounds (Section 8). We show polynomial lower bounds for the maximum matching, triangle counting, and connected components problems in fully dynamic streams. Our lower bounds reduce each of these graph problems via novel graph constructions to answering inner product queries (Dinur and Nissim, 2003; De, 2012; Dwork et al., 2007; Mir et al., 2011). The known lower bounds for inner product queries then apply to our problems. We encode the secret dataset given by the inner product query problem via $\Theta(n)$ insertions to construct our graph. Then, we map the value of the answers to each of

our graph problems to the original inner product query via the inclusion-exclusion principle. Finally, we delete all insertions through edge deletions and repeat the process for the next query. We remark that this technique is quite general and likely extends to many other natural graph problems. Roughly speaking, the only problem-specific requisite is that we can encode the bits of a secret dataset using the problem structure and that we can easily flip the bits by adding and deleting edges. For example, we can encode each bit of a secret dataset as a (non-)edge in an induced matching so that flipping a bit equates to (adding) deleting the (non-)edge.

3. Related Works

Streaming Algorithms. The streaming model of computation is a prominent model for large-scale data analysis that has been studied for multiple decades (Morris, 1978). In this model, one usually seeks space- and time-efficient algorithms that can process the stream on the fly without the need to store all data. A long line of work in this area includes classical results such as the Flajolet-Martin algorithm for counting distinct elements (Flajolet and Martin, 1985) and many others (Flajolet et al., 2007; Alon et al., 1996). In the context of streaming graph algorithms, ideally one would like to obtain space sublinear in the number of vertices (McGregor and Vorotnikova, 2018; Huang and Peng, 2019); but for many graph problems it is necessary to work in the semi-streaming model settling on space near-linear in the number of vertices (Assadi et al., 2022; Ahn and Guha, 2011; Esfandiari et al., 2018; Feigenbaum et al., 2005; Fischer et al., 2022; McGregor et al., 2015). Online streaming algorithms release graph statistics after every update while maintaining the low space bounds (Chen et al., 2023b; Halldórsson et al., 2016; Cormode et al., 2018; Ghosh and Stoeckl, 2024). In the non-private streaming setting, there exist many works on graph algorithms, including approximating the size of the maximum matching (Ahn and Guha, 2011; Assadi et al., 2021; Assadi, 2022; Fischer et al., 2022; McGregor and Vorotnikova, 2018), vertex cover (Assadi et al., 2019), densest subgraph (Esfandiari et al., 2016; McGregor et al., 2015), k-core decomposition (Esfandiari et al., 2018; King et al., 2023), number of connected components (Berenbrink et al., 2014), and others (Assadi et al., 2022; Ahn and Guha, 2011; Feigenbaum et al., 2005; Fischer et al., 2022; Huang and Peng, 2019; Henzinger et al., 1998; Muthukrishnan et al., 2005; McGregor, 2014).

Continual Release Model. In the context of streaming computation, the DP model of reference is the continual release model (Dwork et al., 2010a; Chan et al., 2011) where we require algorithms to abide by a strong privacy notion: an observer obtaining *all* future outputs of the algorithm must in essence learn almost nothing about the existence of any single input. Since its introduction, this research area has received vast attention outside of graphs, including many recent works (see e.g. Chan et al. (2012); Henzinger et al. (2024b, 2023); Fichtenberger et al. (2023); Jain et al. (2023a,b)).

Among the insertion-only continual release work, prior research has tackled classical estimation problems (Chan et al., 2011; Henzinger et al., 2024b, 2023), as well as heavy hitters-related problems (Chan et al., 2012; Epasto et al., 2023). More recent works tackle the fully dynamic continual release setting (Jain et al., 2023a; Dupré la Tour et al., 2023; Fichtenberger et al., 2021). Particularly relevant is (Fichtenberger et al., 2021), which shows hardness results for graph estimation in fully dynamic streams. Our work contributes new hardness results in this new emerging area as well.

Most relevant to our paper is the literature on continual release algorithms in graphs (Upadhyay et al., 2021; Fichtenberger et al., 2021; Song et al., 2018; Jain et al., 2024). In this area, Song et al. (2018) studied graph statistics (degree distribution, subgraph counts, etc.) on bounded-degree graphs. Fichtenberger et al. (2021) focused on estimating a number of graph statistics like maximum matching, triangle counting, and the density of the densest subgraph for both edge and node privacy; they provide approximation guarantees in terms of the maximum degree in the graph as well as lower bounds in insertion-only and fully dynamic streams. Jain et al. (2024) explored counting problems in graphs (counting edges, triangles, stars, connected components) for node-privacy and where privacy must hold for arbitrary graphs (e.g., graphs with arbitrary degrees). They give time-aware projection algorithms that can transform any continual release algorithm that gives approximation guarantees for bounded degree graphs into a truly private algorithm on nearly bounded degree graphs. Like the work of Jain et al. (2024), for our algorithms that assume a public bound, say on the stream arboricity, this bound affects only the approximation guarantees but not the privacy claims, hence our algorithms satisfy their notion of *truly private* (Jain et al., 2024).

Private Graph Algorithms. The private literature on graph algorithms includes a large body of work on static graph algorithms (see e.g. Arora and Upadhyay (2019); Blocki et al. (2022); Dinitz et al. (2025a); Dhulipala et al. (2022); Eden et al. (2023); Kasiviswanathan et al. (2013); Kalemaj et al. (2023); Liu et al. (2024); Mueller et al. (2024); Nissim et al. (2007); Raskhodnikova and Smith (2016a,b); Upadhyay (2013) and references therein). Aside from the problems we study, work in this area includes results on preserving graph cuts (Arora and Upadhyay, 2019), the stochastic block model recovery (Chen et al., 2023a), graph clustering (Bun et al., 2021; Imola et al., 2023), and many other areas.

Our paper focuses on the pure, ε-DP setting. In this setting, there have been a variety of recent works that we study for static graphs. The densest subgraph (DSG) problem has been extensively studied in the non-private streaming context (Bhattacharya et al., 2015; Esfandiari et al., 2016; McGregor et al., 2015). Our work builds on the results of Esfandiari et al. (2016); McGregor et al. (2015), which show that edge sampling approximately preserves the density of the densest subgraph. In the context of privacy, beyond the already cited work of Fichtenberger et al. (2021) that focused on density estimation only, all other works are in the static DP or LEDP setting (Nguyen and Vullikanti, 2021; Farhadi et al., 2022; Dhulipala et al., 2022; Dinitz et al., 2025a; Dhulipala et al., 2023). In the ε-DP setting, the best known lower bound on the additive error of the densest subgraph problem is $\Omega(\sqrt{\varepsilon^{-1}\log n})$ Farhadi et al. (2022); Nguyen and Vullikanti (2021). Comparatively, the best known upper bounds in the central ε-DP setting are the $(2, O(\varepsilon^{-1}\log n))$-approximation (Dhulipala et al., 2023), $(2+\eta, O(\varepsilon^{-1}\log^2 n))$-approximation (Dinitz et al., 2025a), and the $(1+\eta, O(\varepsilon^{-1}\log^4 n))$-approximation (Dhulipala et al., 2022) upper bounds. In the ε-LEDP setting, the best known upper bounds are the $(2+\eta, O(\eta^{-1}\log^2 n))$-approximate (Dinitz et al., 2025a) and $(2, O(\varepsilon^{-1}\log n))$-approximate (Dhulipala et al., 2023) bounds. Our paper presents the first continual release algorithm for releasing an actual approximate densest subgraph (as opposed to estimating its density) in edge-private insertion-only streams.

Related to the densest subgraph problem is the k-core decomposition of the graph for which some streaming results are known (Esfandiari et al., 2018; King et al., 2023; Sariyüce et al., 2013). In the private setting, this problem has been tackled by Dhulipala et al. (2022, 2023) and Henzinger et al. (2024a), where the best error bound achieved in both

the central ε-DP and ε-LEDP settings provides $(1, O(\varepsilon^{-1} \log n))$-approximations of the core number (Dhulipala et al., 2023). This is tight against the recently shown lower bound of $\Omega(\varepsilon^{-1} \log n)$ (Henzinger et al., 2024a). In our paper, we provide the first continual release algorithm for approximating the core number of nodes in a graph.

3.1. **Concurrent Work.** Recent concurrent work of Raskhodnikova and Steiner (2025) studies edge-DP algorithms in the fully dynamic continual release model. They give a number of upper and lower bounds for a variety of problems, including triangle counting, connected component counting, maximum matching size, and degree histogram. They consider *event-level* and *item-level* privacy. *Event-level* describes fully dynamic neighboring streams where one update differs, while *item-level* describes neighboring streams where all updates on a particular edge differ. Note that for insertion-only streams with no duplicate insertions, both settings describe the same set of neighboring streams. They give the first $\text{poly}(T, n)$ error lower bounds for (ε, δ)-DP graph algorithms in continual release for $\delta > 0$; previous works only considered $\delta = 0$ in their lower bounds. Their item-level upper bounds are tight against their lower bounds, with additive error proportional to $\text{poly}(T, n)$.

Similar to us, their lower bounds are also based on batch-to-continual reductions to known static DP lower bounds. On the other hand, our work considers different problems (densest subgraph and k-core decomposition) and focuses on designing algorithms with $\text{poly}(\log n)$ additive errors in insertion-only streams. Raskhodnikova and Steiner (2025) focus on fully dynamic streams, which in general require additive $O(\text{poly}(T, n))$ error, as both their lower bounds and ours (Section 8) demonstrate. Furthermore, our paper focuses on the sublinear space setting and returning vertex subset solutions, two settings not discussed in (Raskhodnikova and Steiner, 2025).

3.2. **Subsequent Work.** The subsequent work of Zhou (2026) improved on the additive error and the space complexity of our densest subgraph algorithm by logarithmic factors. Specifically, through the use of graph *densification*, they obtain a continual release densest subgraph algorithm whose additive error matches the best known static DP algorithms up to constant factors and whose space complexity matches the best known streaming algorithms up to constant factors.

Dinitz et al. (2025b) designed a continual release matching algorithm that returns implicit matching solutions, as returning explicit edge subsets violates privacy. However, their algorithm does not ensure sublinear space for dense graphs.

4. Preliminaries

We now introduce the notation we use in this paper as well as some definitions. Further standard preliminaries are deferred to Appendix A.

4.1. **Setting & Notation.** A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E where edge $\{u, v\} \in E$ if and only if there is an edge between $u \in V$ and $v \in V$. We write $n := |V|$ and $m := |E|$.

We consider the insertion-only setting within the continual release model, where we begin with an empty graph $G_0 = (V, E_0)$ where $E_0 = \varnothing$ and edge updates in the form of $\{e_t, \perp\}$ arrive in a stream at every timestep $t \in [T]$ where each non-empty edge is inserted at most once. We are required to release an output for the problem of interest after every (possibly empty) update. Our goal is to obtain algorithms that achieve sublinear space, either in the number of edges $\tilde{o}(m)$ (note that $T \geq m$ as we allow for empty updates), or the number of vertices $\tilde{o}(n)$. We assume $T \leq n^c$ for some absolute constant $c \geq 1$ to simplify our presentation. If there are no empty edge updates then it is sufficient to consider $c = 2$.

Throughout the paper, we write $\eta \in (0, 1]$ to denote a fixed multiplicative approximation parameter.

We use $\mathcal{M}$ to denote a mechanism, $M(\cdot)$ to denote the size of a maximum matching, and μ to denote the mean of a random variable.

4.2. **Continual Release.** We use the phrasing of the below definitions as given by Jain et al. (2024).

Definition 4.1 (Graph Stream (Jain et al., 2024))**.** In the continual release model, a graph stream $S \in \mathcal{S}^T$ of length T is a T-element vector where the i-th element is an edge update $u_i = \{v, w, insert\}$ indicating an edge insertion of edge $\{v, w\}$, $u_i = \{v, w, delete\}$ indicating an edge deletion of edge $\{v, w\}$, or $\perp$ (an empty operation).

We use G_t and E_t to denote the graph induced by the set of updates in the stream S up to and including update t. Now, we define neighboring streams as follows. Intuitively, two graph streams are edge neighbors if one can be obtained from the other by removing one edge update (replacing the edge update by an empty update in a single timestep).

Definition 4.2 (Edge Neighboring Streams)**.** Two streams of edge updates, $S = [u_1, \ldots, u_T]$ and $S' = [u'_1, \ldots, u'_T]$, are *edge-neighboring* if there exists exactly one timestamp $t^* \in [T]$ where $u_{t^*} \neq u'_{t^*}$ and for all $t \neq t^* \in [T]$, it holds that $u_t = u'_t$. Streams may contain any number of empty updates, i.e. $u_t = \perp$. Without loss of generality, we assume for the updates u_{t^*} and u'_{t^*} that $u'_{t^*} = \perp$ and $u_{t^*} = \pm e_{t^*}$ is an edge insertion or deletion.

The notion of neighboring streams leads to the following definition of an edge differentially private algorithm.

Definition 4.3 (Edge Differential Privacy)**.** Let $\varepsilon, \delta \in (0, 1)$. An algorithm $\mathcal{A}(S) : \mathcal{S}^T \to \mathcal{Y}^T$ that takes as input a graph stream $S \in \mathcal{S}^T$ is said to be (ε, δ)-*edge differentially private (DP)* if for any pair of edge-neighboring graph streams S, S' that differ by 1 edge update and for every T-sized vector of outcomes $Y \subseteq \text{Range}(\mathcal{A})$,

$$\mathbf{P}\left[\mathcal{A}(S) \in Y\right] \leq e^{\varepsilon} \cdot \mathbf{P}\left[\mathcal{A}(S') \in Y\right] + \delta.$$

When $\delta = 0$, we say that $\mathcal{A}$ is ε-*edge DP*.

Note that all of our algorithms handle batched edge updates where u_i can be a set of edge updates and the multiset of all updates differs by one edge. For each batch, we can simply execute the non-batched algorithm on each update in the batch and return the output after the final update in the batch.

We now formalize the concept of edge edit distance between streams and the concept of *stability* under sparsification.

Definition 4.4 (Edge Edit Distance)**.** Given two streams $S, S' \in \mathcal{S}^T$, the edge edit distance between the two streams is the shortest chain of graph streams $S_0, S_1, \ldots, S_d$ where $S_0 = S$ and $S_d = S'$ where every adjacent pair of streams in the chain are edge-neighboring. The edge edit distance is d, the length of the chain.

An algorithm that takes as input a stream S and outputs a chosen set of updates from the stream is *stable* if on edge-neighboring streams S and S', there exists a coupling[7] between the randomness used in the algorithm on the inputs such that the edge edit distance between the output streams is $O(1)$.

5. k-Core Decomposition

In this section, we introduce a semi-streaming sampling algorithm that preserves the k-cores in an input graph $G = (V, E)$ while ensuring privacy. Specifically, we have the following theorem.

Theorem 5.1 (Sublinear Space Private k-Core Decomposition)**.** *Fix $\eta \in (0, 1]$. Algorithm 5.2 is an ε-DP algorithm for the k-core decomposition problem in the continual release model for insertion-only streams. At every $t \in [T]$, the algorithm returns a value for each vertex, such that every value is a $\left(2+\eta, O\left(\eta^{-2}\varepsilon^{-1}\log^3(n)\right)\right)$-approximation of the corresponding vertex's true core value, with probability $1 - 1/\operatorname{poly}(n)$. The maximum space used is $O\left(\eta^{-4}\varepsilon^{-1} n \log^5 n\right)$, with probability $1 - 1/\operatorname{poly}(n)$.*

The k-core decomposition algorithm we give in this paper also translates into a novel k-core decomposition algorithm in the non-private setting. We give this non-private algorithm in the appendix along with the associated proofs in Appendix B.

Algorithm Intuition. First, we give some intuition for our algorithm (Algorithm 5.2). Our algorithm essentially performs a sampling version of the classic peeling algorithm for k-core decomposition. The classic peeling algorithm successively *peels* (removes) vertices with the minimum degree until all vertices are removed from the graph. A core that is formed during the peeling process is the induced subgraph consisting of the remaining vertices after a vertex is peeled and the value of such a core is the minimum induced degree within the subgraph. The core number for each vertex v is equal to the maximum valued core that v is a part of during any stage of the peeling. A dynamic version of this algorithm can be obtained by maintaining a *level data structure*, where the levels are iteratively updated and initialized to be all the same. A vertex is moved up a level if its induced degree among vertices in the same or higher levels is larger than a cutoff C. One can show that having $O(\log n)$ levels of the structure and appropriately setting C among $O(\log n)$ duplicates of the structure gives a $(2+\eta)$-approximation of the core numbers of the nodes in the non-private, insertion-only setting (Sun et al., 2020; Liu et al., 2022; Dhulipala et al., 2022). Our main innovation is a private and sparse level data structure.

When sparsifying the graph, we cannot simply take a uniform sample of the edges adjacent to each vertex. An easy example to consider is a vertex v which is part of a

[7] A *coupling* of random variables (X, Y) is a joint distribution such that the marginal distributions correspond to X, Y respectively.

10-clique and also adjacent to $n/2$ degree one vertices. A uniform sample of the edges adjacent to v will most likely not discover the 9 edges connecting it to the 10-clique (for large n). We call the edges connecting v to the 10-clique the set of *important* edges. Thus, we must take a smarter sample of edges adjacent to v. To maintain a sparsified, sampling-based version of the level data structure, we maintain samples of large enough size of the *up-edges* adjacent to each vertex. The up-edges adjacent to each vertex are the edges connecting each vertex to neighbors in the same or higher levels. Once we see enough sampled up-edges, we move the vertex up one level and continue sampling edges until we either reach the topmost level or the vertex is adjacent to only a very small sample of up-edges. Such a sampling method allows us to keep enough of the important edges which connect to other vertices in higher valued cores.

Finally, to make the above algorithm differentially private, we use SVT to determine when to move the vertex up a level. We show that although many vertices may move up levels, we only lose privacy when the vertices that are adjacent to the edge that differs between neighboring streams move up. Since our total number of levels and duplicates is bounded by $O(\log^2 n)$, the privacy loss from SVT is also bounded by $O(\log^2 n)$.

Pseudocode. Our Algorithm 5.1 is inspired by the insertion-only sketching algorithm of Esfandiari et al. (2018) but adapted to the level data structure (Dhulipala et al., 2022); our level data structure sparsification is simpler in nature and uses an entirely new analysis. Furthermore, since the degeneracy of the graph is equal to the maximum core number of any node in the input graph, our algorithm also gives a private approximation of the degeneracy of the input graph in the continual release model.

Analysis. We provide the pseudocode and proof of Theorem 5.1 in Appendix C. Specifically, the pseudocode of the sparse level data structure is provided in Algorithm 5.1 and the pseudocode of the actual algorithm is in Algorithm 5.2. Section 5.1 then presents a detailed description of the algorithm and we prove its privacy and utility guarantees in Appendices C.1 and C.2, respectively.

5.1. **Detailed Algorithm Description.** We now give the detailed description of our algorithm. Our algorithm proceeds as follows. We maintain $O\left(\log_{(1+\eta)}(n)\right)$ subgraphs where for each subgraph, the probability we use to sample edges into the subgraph is different. Let $Q = [\lceil \log_{1+\eta}(L) \rceil, \ldots, \lceil 2\log_{(1+\eta)}(n) \rceil]$. Specifically, we consider all subgraphs $j \in Q$ (Algorithm 5.1, Line 2) where the integer $j \geq \log_{(1+\eta)}(L)$ and we set $L = \frac{c_3 \log^3(n)}{\varepsilon}$ (Algorithm 5.2, Line 25).[8] For each update we receive in the stream (Algorithm 5.2, Line 28), we first determine which subgraphs to sample the edge into using the procedure PrivateCore.SampleEdge(e_t) (Algorithm 5.2, Line 30). The procedure determines whether to sample an edge by iterating through all of the subgraphs X_j for $j \in Q$ (Algorithm 5.1, Line 11). We decide to sample $e_t = \{u, v\}$ into X_j by looking at both endpoints of the edge update (Algorithm 5.1, Line 12). If for either endpoint $w \in \{u, v\}$, vertex w is not on level $F - 1$ of X_j (Algorithm 5.1, Line 13), then we sample the edge using probability p_j (Algorithm 5.1, Line 14).

[8] L represents our base threshold for a core value; its usage ensures our estimates do not evaluate small core values which would cost too much privacy.

Algorithm 5.1: Data Structure for k-Core Decomposition for Adaptive Insertion-Only Streams

```
1  Class PrivateCore(ε, η, L, n, T)
2      Maintain sampled edges X_j ← ∅ for each j ∈ Q where
         Q = {⌈log_(1+η)(L)⌉, ..., ⌈2 log_(1+η)(n)⌉}
3      Initialize F ← ⌈2 log_(1+η)(n)⌉        (F − 1 denotes the topmost level of any level
         data structure)
4      Initialize ε_1 ← ε/(6|Q|F)
5      for j ∈ Q do
6          Initialize p_j ← c_1 log(n) / (ε_1 (1+η)^j)
7          for v ∈ V do
8              Initialize class SVT^{j,v}(ε_1, 1, 1)        (Algorithm A.1)
9              Initialize levels[j][v] ← 0
10     Function SampleEdge(e_t)
11         for j ∈ Q do
12             for w ∈ e_t = {u, v} do
13                 if levels[j][w] < F − 1 then
14                     Sample e_t into X_j with probability p_j
15     Function UpdateLevels()
16         for level ℓ ∈ {0, ..., F − 2} do
17             for j ∈ Q do
18                 for v ∈ V do
19                     if levels[j][v] ≠ ℓ then
20                         go to next iteration
21                     if SVT^{j,v}.ProcessQuery(deg^+_{X_j}(v), p_j · (1 + η)^{j−1}) is "above"
                         then
22                         levels[j][v] ← levels[j][v] + 1
23     Function GetPrivateLevels()
24         return levels
```

Each X_j is organized into levels where vertices are moved up levels if they have induced degree among vertices at the same or higher level *approximately* greater than $(p_j) \cdot (1+\eta)^{j-1}$. We require the degree to be approximately greater rather than exactly greater to preserve privacy.

After sampling the new edge e_t, we then perform PrivateCore.UpdateLevels() (Algorithm 5.2, Line 31) which updates the levels of each vertex. Within the procedure, for each sampled graph X_j (Algorithm 5.1, Line 17) and for each level ℓ starting from the bottommost level and iterating to the top level (Algorithm 5.1, Line 16), we check each vertex $v \in V$ (Algorithm 5.1, Line 18) to determine whether we need to move that vertex up a level.

Algorithm 5.2: Algorithm for k-Core Decomposition for Adaptive Insertion-Only Streams

```
Initialize initial core threshold L ← c3 log^3(n) / ε
Initialize PrivateCore(ε, η, L)
levels ← PrivateCore.GetPrivateLevels()
for each new update e_t do
    if e_t ≠ ⊥ then
        PrivateCore.SampleEdge(e_t)
    PrivateCore.UpdateLevels()
    levels ← PrivateCore.GetPrivateLevels()
    for every vertex v ∈ V do
        j_now ← max{j : levels[j][v] = F − 1}
        if (1 + η)^{j_now} > L then
            Release v's core as (2 + η) · (1 + η)^{j_now}
        else
            Release v's core as 1
```

Let $\deg^+_{X_j}(v)$ be the degree of v in the induced subgraph of its neighbors at the same level or higher. To see whether we should move that vertex up a level, we first determine whether the current level of that vertex is the same level that we are iterating over (Algorithm 5.1, Line 19). If it is the same level and we pass the SVT check that $\deg^+_{X_j}(v)$ exceeds $p_j \cdot (1+\eta)^{j-1}$ (Algorithm 5.1, Line 21), then we move the vertex up a level by incrementing $levels[j][v]$ (Algorithm 5.1, Line 22). The particular threshold of $p_j \cdot (1+\eta)^{j-1}$ that we use will become apparent once we discuss our analysis of the approximation factor. Intuitively, the levels mimic the traditional peeling algorithm for the static k-core decomposition problem. The threshold $(1+\eta)^{j-1}$ is used in previous works (e.g. (Dhulipala et al., 2022)) and because we are sampling edges instead of using the entire graph, the threshold is multiplied by p_j.

Finally, in the last part of the algorithm, we iterate through each vertex (Algorithm 5.2, Line 33) and release an estimate if the vertex v is in the topmost level of a subgraph $X_{j_{now}}$ where j_{now} is maximized. If this is the case, we release the new estimate $(2+\eta) \cdot (1+\eta)^{j_{now}}$ for vertex v (Algorithm 5.2, Line 36), assuming $(1+\eta)^{j_{now}}$ exceeds some data-oblivious lower bound L.

6. Densest Subgraph

In this section, we focus on the densest subgraph problem and provide the first differentially private algorithm for densest subgraph in the continual release model using space sublinear in the total number of edges in the graph.

Theorem 6.1 (Sublinear Space Private Densest Subgraph)**.** *Fix $\eta \in (0, 1]$. Algorithm 6.2 is an ε-edge differentially private algorithm for the densest subgraph problem in the continual release model for insertion-only streams. The algorithm returns a set of vertices whose induced subgraph is a $\left(2+\eta, O(\eta^{-1}\varepsilon^{-1}\log^2(n))\right)$-approximation of the densest subgraph in*

G_t, with probability at least $1 - 1/\operatorname{poly}(n)$, for all $t \in [T]$. The maximum space used is $O\left(\eta^{-2}\varepsilon^{-1} n \log^2(n)\right)$, with probability at least $1 - 1/\operatorname{poly}(n)$, for all $t \in [T]$.

We can also reduce the multiplicative error to $(1+\eta)$ at the cost of increasing the space usage by a $\operatorname{poly}(\log(n))$ factor.

Theorem 6.2 (Sublinear Space Private Densest Subgraph)**.** *Fix $\eta \in (0, 1]$. There exists an ε-edge differentially private algorithm for the densest subgraph problem in the continual release model for insertion-only streams. The algorithm returns a set of vertices whose induced subgraph is a $\left(1+\eta, O(\eta^{-4}\varepsilon^{-1}\log^5(n))\right)$-approximation of the densest subgraph in G_t, with probability at least $1 - 1/\operatorname{poly}(n)$, for all $t \in [T]$. The maximum space used is $O\left(\eta^{-5}\varepsilon^{-1} n \log^5(n)\right)$, with probability at least $1 - 1/\operatorname{poly}(n)$, for all $t \in [T]$.*

Algorithm Intuition. We revise the algorithms of McGregor et al. (2015) and Esfandiari et al. (2016) to the insertion-only continual release setting. On a high level, our algorithm maintains a sample of edges over time and releases a DP set of vertices by running a black-box DP densest subgraph algorithm (e.g., Theorem G.2) on the subgraph induced by the sample. At every timestep $t \in [T]$, an edge e_t is sampled with probability p — the sampling probability is initialized to 1 as in the beginning we can afford to store every edge. We privately check whether the number of edges seen so far exceeds a certain threshold using a sparse vector technique (SVT) query and adjust the sampling probability p accordingly. In order to avoid privacy loss that grows linearly in T, we do not invoke the black-box DP densest subgraph algorithm at every timestep and instead invoke it only if the current density of the sample exceeds a certain threshold using another SVT query.

Analysis. The proofs of Theorems 6.1 and 6.2 are deferred to Appendix D. We present the pseudocode for a useful data structure in Algorithm 6.1 and the main algorithm is in Algorithm 6.2. Section 6.1 describes the algorithm in detail and we prove its privacy and utility guarantees in Appendices D.1 and D.2, respectively.

6.1. **Detailed Algorithm Description.** We first define a data structure `PrivateDSG` (see Algorithm 6.1) for maintaining an approximate densest subgraph in insertion-only streams. This data structure takes as input an accuracy parameter η, the number of nodes n, and a bound on the stream size T and uses as a black-box PrivateDensestSubgraph (an ε-DP static densest subgraph algorithm) and Densest-Subgraph (a non-private static densest subgraph algorithm). This data structure has three procedures: an update procedure (`SampleEdge`), a procedure for getting a private densest subgraph (`GetPrivateApproxDensestSubgraph`), and a procedure for returning the non-private exact density of the sampled subgraph (`GetNonPrivateDensity`).

Our main algorithm (see Algorithm 6.2) uses the data structure above and performs its initialization based on the input accuracy parameter η, the number of nodes n, the privacy parameter ε, and an upper bound T on the total number of updates in the stream.

We perform the following initializations:

(i) an instance of PrivateDSG(ε, η, n, T) (Algorithm 6.2, Line 27),
(ii) a counter for the number of edge insertions we have seen so far in our stream (Algorithm 6.2, Line 25),
(iii) the privacy parameters for the different parts of our algorithm (Algorithm 6.2, Line 26),

Algorithm 6.1: Data Structure for Densest Subgraph in Adaptive Insertion-Only Streams

Class `PrivateDSG(`ε, η, n, T`)`
- Maintain sampled edges $X \leftarrow \varnothing$
- Set $m' \leftarrow \frac{c_3 n \log^2(n)}{\varepsilon\eta^2}$ using a large enough constant $c_3 > 0$
- Set $p \leftarrow 1$
- Initialize empty sampling hashmap H to store edges and their associated random weight
- Initialize class $\mathrm{SVT}(\varepsilon, 1, c_5 \log_{1+\eta}(n))$ (Algorithm A.1)
- **Function** `SampleEdge(`e_t, m, ε`)`
  - **if** $\mathrm{SVT.ProcessQuery}(m, m')$ *is "above"* **then**
    - $m' \leftarrow (1+\eta) \cdot m'$
    - $p \leftarrow \min\left(1, \frac{c_4 n \log^2(n)}{\varepsilon \cdot \eta^2 m'}\right)$
    - **for** *each edge* $e \in H$ **do**
      - **if** $H[e] \leq p$ **then**
        - Keep e in X
      - **else**
        - Remove e from X and H
  - **if** $e_t \neq \perp$ **then**
    - Sample $h_{e_t} \sim U[0,1]$ uniformly at random in $[0,1]$
    - **if** $h_{e_t} \leq p$ **then**
      - Store $X \leftarrow X \cup \{e_t\}$
      - Add $H[e_t] = h_{e_t}$
- **Function** `GetPrivateApproxDensestSubgraph(`ε`)`
  - Return $\mathrm{PrivateDensestSubgraph}(\varepsilon, X)$ (Theorem G.1)
- **Function** `GetNonPrivateDensity()`
  - Return $(\mathrm{ExactDensity}(X), p)$ (Theorem H.1)

(iv) the initial density threshold for the density for returning a new private densest subgraph solution (Algorithm 6.2, Line 28) where $c_1 > 0$ is a constant.

The initial cutoff for the density is equal to our additive error since we can just return the entire set of vertices as long as the density of the densest subgraph is (approximately) less than our additive error.

We now receive an online stream of updates (which can be empty, $\perp$) one by one. The t-th update is denoted e_t (Algorithm 6.2, Line 31). For each update, we first check whether the update is an edge insertion or $\perp$ (Algorithm 6.2, Line 32); if it is not $\perp$, then we increment m (Algorithm 6.2, Line 33). We then call $\mathrm{PrivateDSG.SampleEdge}(e_t, m, \varepsilon_2)$ which decides how to sample the edge (Algorithm 6.2, Line 34).

The SampleEdge procedure is within the PrivateDSG class which maintains the following:

(i) a set X (Algorithm 6.1, Line 2) of sampled edges in the stream,

Algorithm 6.2: Algorithm for Densest Subgraph in Adaptive Insertion-Only Streams

25 Initialize counter of number of edges $m \leftarrow 0$
26 Initialize privacy parameters $\varepsilon_1 \leftarrow \frac{\varepsilon}{3c_2 \log_{1+\eta}(n)}, \varepsilon_2 \leftarrow \varepsilon/3$
27 Initialize PrivateDSG$(\varepsilon_1, \eta, n, T)$
28 Initialize density threshold $L \leftarrow \frac{(1+\eta)c_1 \cdot \log^2(n)}{\varepsilon\eta^2}$
29 Initialize class SVT$(\varepsilon_2, 1, c_2 \log_{1+\eta}(n))$ (Algorithm A.1)
30 $S \leftarrow V$
31 **for** *each new update* e_t **do**
32 **if** $e_t \neq \perp$ **then**
33 $m \leftarrow m + 1$
34 PrivateDSG.SampleEdge(e_t, m, ε_2)
35 $D, p \leftarrow$ PrivateDSG.GetNonPrivateDensity()
36 **if** SVT.ProcessQuery$(D, p \cdot L)$ *is "above"* **then**
37 $L \leftarrow (1 + \eta) \cdot L$
38 $S \leftarrow$ PrivateDSG.GetPrivateApproxDensestSubgraph(ε_1)
39 Release S

(ii) an estimate, m', of the number of edges seen so far in the stream (Algorithm 6.1, Line 3),
(iii) the probability p by which the current edge in the stream is sampled (Algorithm 6.1, Line 4).

The initial probability that we sample an edge is 1 since we have not seen many edges and can keep all of them within our space bounds.

In order to sample according to the desired (private) probability, we first privately check whether the current number of edges we have seen exceeds the threshold m' using the *sparse vector technique* (Algorithm A.1). Our sparse vector technique function (Algorithm 6.1, Line 6) is initialized with

(i) the privacy parameter ε_2,
(ii) the sensitivity of the query (which is 1 in our setting),
(iii) the maximum number of successful queries (we perform queries until we've reached the end of the stream or we exceed the successful queries threshold).

Then, the SVT query is run with the query, m, (which is the current number of edges we've seen so far) and the threshold, m' (Algorithm 6.1, Line 8). If the SVT query passes, then we update m' to a larger threshold (Algorithm 6.1, Line 9) and also update the probability of sampling in terms of the new m' (Algorithm 6.1, Line 10) (using constant c_4). We then sample the edges by sampling a value h_{e_t} uniformly from $[0, 1]$ (Algorithm 6.1, Line 17). If $h_{e_t} \leq p$ (Algorithm 6.1, Line 18), then we store e_t in X (Algorithm 6.1, Line 19) and the value h_{e_t} in our hashmap H (Algorithm 6.1, Line 20). Whenever p changes, we also have to resample the edges in X; to do this, we keep the edge in X if $H[e_t] \leq p$ and remove e_t otherwise (Algorithms 6.1, Line 12, 6.1, Line 13 and 6.1, Line 15). This ensures that the marginal probability of sampling an edge at any timestamp t is p.

After we have sampled our edges, we now need to obtain the *non-private* density of our current subgraph (Algorithm 6.2, Line 35). If the non-private density exceeds our density threshold L via private SVT (Algorithm 6.2, Line 36), then we increase our threshold (Algorithm 6.2, Line 37) and use our private densest subgraph algorithm to return a private approximate densest subgraph (Algorithm 6.2, Line 38). We release our stored private graph (Algorithm 6.2, Line 39) for all new updates until we need to compute a new private densest subgraph. The densest subgraph algorithm that we use to obtain a differentially private densest subgraph can be any existing static edge-DP algorithm for densest subgraph like Theorem G.1 which gives a $(2, O(\varepsilon_1^{-1}\log(n)))$-approximation.

7. Maximum Matching

In this section we give an edge-DP algorithm for maximum matching that uses truly sublinear space. We adapt the algorithm of McGregor and Vorotnikova (2018) to obtain a private approximate maximum cardinality matching algorithm in the continual release model using sublinear space in the number of vertices. As in their algorithm, we assume that we are provided with a public upper bound $\tilde{\alpha}$ on the maximum arboricity α of the graph at any point in the stream. We also briefly sketch how to remove the assumption on $\tilde{\alpha}$ in Appendix I.2 at the cost of more space and a worse approximation guarantee.

The privacy of our algorithm is always guaranteed. However, we do not assume that $\tilde{\alpha}$ is guaranteed to upper bound α. When $\tilde{\alpha} \geq \alpha$, our approximation guarantees hold with high probability. Otherwise, our approximation guarantees do not necessarily hold. Note that the same type of guarantee holds for the original non-private streaming algorithm (McGregor and Vorotnikova, 2018) where their utility guarantee is only given when their public estimate of α upper bounds the maximum arboricity of the input. We prove the following result in this section.

Theorem 7.1. *Fix $\eta \in (0, 1]$. Given a public estimate $\tilde{\alpha}$ of the maximum arboricity α over the stream,[9] Algorithm 7.1 is an ε-edge DP algorithm for estimating the size of the maximum matching in the continual release model for insertion-only streams. If $\tilde{\alpha} \geq \alpha$, then with probability at least $1 - 1/\operatorname{poly}(n)$, our algorithm returns a $\big((1+\eta)(2+\tilde{\alpha}), O(\eta^{-1}\varepsilon^{-1}\log^2(n))\big)$-approximation of the size of the maximum matching at every timestamp. Moreover, our algorithm uses $O(\eta^{-2}\varepsilon^{-1}\log^2(n)\log(\tilde{\alpha}))$ space with probability $1 - 1/\operatorname{poly}(n)$.*

Algorithm Intuition. We revise the algorithm of McGregor and Vorotnikova (2018) to the insertion-only continual release setting. On a high level, McGregor and Vorotnikova (2018) showed that the cardinality of a carefully chosen subset of edges $F \subseteq E$ is a good estimator for the size of the maximum matching for graphs of bounded arboricity α. F is obtained from E by deleting edges e adjacent to a vertex v if more than α other edges adjacent to v arrived after e. Then their algorithm maintains a small sample $S \subseteq F$ throughout the algorithm by down-sampling edges when the current sample exceeds some threshold. Our algorithm follows a similar approach with two main adjustments to satisfy privacy. First, we release powers of $(1+\eta)$ based on an SVT comparison against the current value of the estimator. Second, the decision to down-sample is also based on an SVT comparison against the threshold.

[9] We can eliminate this assumption with an additional pass of the stream or notify the observer when the utility guarantees no longer hold in the one-pass setting. See Appendix I.

Analysis. The proof of Theorem 7.1 is deferred to Appendix E. The pseudocode for handling edge updates is presented in Algorithm 7.1. Then, Section 7.1 provides a detailed description of the algorithm. We prove its privacy and utility guarantees in Appendices E.1 and E.2, respectively.

Algorithm 7.1: Sampling E_t^*

1 **Class** `EdgePrivateSublinearMatching`$(\tilde{\alpha}, \varepsilon, \eta, n, T)$
2 $S \leftarrow \varnothing$
3 $p_1 \leftarrow 1$
4 $Q_1 \leftarrow a_1 \log(n), Q_2 \leftarrow a_2 \eta^{-1} \log(n)$
5 $j_1 \leftarrow 0$
6 estimate $\leftarrow (1+\eta)^{j_1}$
7 Initialize class SubsampleSVT $\leftarrow$ SVT$(\varepsilon/2, 2, Q_1)$ (Algorithm A.1)
8 Initialize class EstimateSVT $\leftarrow$ SVT$(\varepsilon/2, 2, Q_2)$
9 **Function** `ProcessUpdate`(e_t)
10 **if** $e_t \neq \perp$ **then**
11 With probability p_t add e_t to S and initialize counters $c_e^u \leftarrow 0$ and $c_e^v \leftarrow 0$
12 **for** *each edge* $e' \neq e_t \in S$ **do**
13 (even if e_t is not sampled, it affects the counters of other edges)
14 **if** $e' = \{w, w'\}$ *shares endpoint* w *with* e_t **then**
15 Increment $c_{e'}^w$
16 **if** $c_{e'}^w > \tilde{\alpha}$ **then**
17 Remove e' from S and delete $c_{e'}^w, c_{e'}^{w'}$
18 **while** EstimateSVT.ProcessQuery$(|S|, p_t \cdot (1+\eta)^{j_t})$ *is "above"* **do**
19 $j_t \leftarrow j_t + 1$
20 estimate $\leftarrow (1+\eta)^{j_t}$
21 Output estimate
22 $p_{t+1} \leftarrow p_t$, $j_{t+1} \leftarrow j_t$
23 **if** SubsampleSVT.ProcessQuery$(|S|, \frac{a_3 \log^2(n)}{\varepsilon \eta^2})$ *is "above"* **then**
24 $p_{t+1} \leftarrow p_t/2$
25 **for** $e = \{u, v\} \in S$ **do**
26 With probability $1/2$ remove e from S and delete c_e^u, c_e^v

7.1. **Detailed Algorithm Description.** The pseudocode for our algorithm is given in Algorithm 7.1. Our algorithm works as follows. We set the probability of sampling edges from the stream initially to 1. For each update e_t, we sample the update into S with probability p_t (Algorithm 7.1, Line 11). If the sampled update is not $\perp$ (Algorithm 7.1, Line 10), then we initialize counters for its endpoints if one or both of these counters do not already exist (Algorithm 7.1, Line 11). A sampled e_t that is $\perp$ does not increase the size of S (it is a no-op).

Then, we iterate through every other edge in S (Algorithm 7.1, Line 12) and if e' shares an endpoint with e_t (Algorithm 7.1, Line 14), then we increment the counter $c^w_{e'}$ associated with e' (Algorithm 7.1, Line 15). If the counter exceeds our cutoff $\tilde{\alpha}$ (Algorithm 7.1, Line 16), then we remove e' and its corresponding counters from S (Algorithm 7.1, Line 17). After performing our edge sampling, we now check using SVT whether $|S|$ (the number of edges in S) exceeds $\frac{a_3 \log^2 n}{\varepsilon\eta^2}$ for some large enough constant $a_3 > 0$ (Algorithm 7.1, Line 23). If the SVT AboveThreshold query is satisfied, then we halve the probability of sampling (Algorithm 7.1, Line 24) and resample each edge in S with $1/2$ probability (Algorithm 7.1, Line 26). Finally, we output a new estimate if $|S|/p$ is greater than our previous estimate by a significantly large amount. Specifically, we check via SVT queries (Algorithm 7.1, Line 18) whether $|S|$ exceeds $p \cdot (1+\eta)^{j_t}$. If so, we increment the counter j_t and output our new estimate as $(1+\eta)^{j_t}$ (Algorithm 7.1, Line 20); otherwise, we output our old estimate.

8. Lower Bounds for Fully Dynamic Streams

In this section, we establish lower bounds on the additive error of differentially private algorithms for estimating the size of a maximum matching, the number of triangles, and the number of connected components in the continual release model. Similar to the lower bound for counting distinct elements in the continual release model (Jain et al., 2023a), we reduce the problem of answering matching queries, triangle counting queries, and connected component queries to answering *inner product queries*. Then, we leverage known lower bounds for the inner product problem (Dinur and Nissim, 2003; Dwork et al., 2007; Mir et al., 2011; De, 2012) to obtain our lower bounds.

Theorem 8.1. *Fix $\varepsilon \in (0,1)$. If $\mathcal{A}$ is an ε-DP mechanism that answers maximum matching queries on graphs with n vertices in the continual release model within additive error ζ with probability at least 0.99 for fully dynamic streams of length T, then*

$$\zeta = \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right).$$

Moreover, we may assume the graph is bipartite, has maximum degree 2, and arboricity 1.

Theorem 8.2. *Fix $\varepsilon \in (0,1)$. If $\mathcal{A}$ is an ε-DP mechanism that answers triangle counting queries on graphs with n vertices in the continual release model within additive error ζ with probability at least 0.99 for fully dynamic streams of length T, then*

$$\zeta = \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right).$$

Moreover, we may assume the graph has maximum degree 2 and arboricity 2.

Theorem 8.3. *Fix $\varepsilon \in (0,1)$. If $\mathcal{A}$ is an ε-DP mechanism that answers connected component queries on graphs with n vertices in the continual release model within additive error ζ with probability at least 0.99 for fully dynamic streams of length T, then*

$$\zeta = \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right).$$

Moreover, we may assume the graph is bipartite, has maximum degree 2, and arboricity 2.

Remark 8.4. Theorems 8.1 to 8.3 are proven for pure ε-DP mechanisms, but the argument can be extended to (ε, δ)-DP mechanisms for any $\delta = o(1/T)$. This follows from the fact that our reductions (see Appendix F) use group privacy, which applies equally to approximate DP (Jain et al., 2023a).

The proofs of Theorems 8.1 to 8.3 are presented in Appendix F. In Appendix F.1, we review the key problem which we reduce to matching, triangle counting, and connected components. The lower bound for maximum matching is proven in Appendix F.2, the lower bound for triangle counting in Appendix F.3, and similarly for connected components in Appendix F.4.

8.1. **Further Graph Statistics.** The underlying idea for the maximum matching, triangle counting, and connected components lower bounds (Theorem 8.1, Theorem 8.2, Theorem 8.3) is that we can encode the bits of a secret database $y \in \{0,1\}^n$ within the structure of a sparse graph. Privately answering an inner product query on this database then reduces to answering a "bitwise OR" query by the inclusion-exclusion principle.

It is not hard to see that this technique extends to k-edge-connected component queries and k-vertex-connected component queries.

9. Conclusion & Future Work

In this paper, we initiated the study of low-space continual release algorithms for general graph problems. Using techniques from the non-private graph sparsification literature, we provided continual release algorithms for a variety of general graph problems that, for the first time, achieve nearly the same space and approximation guarantees as their non-private streaming counterparts. The improved space bounds are especially relevant for enabling computations on massive datasets, which are the core motivation of the field of online streaming algorithms.

For our upper bounds, we focused on the insertion-only setting of continual release. As the area of fully-dynamic algorithms in the continual release model is largely unexplored, we believe that an interesting future research direction is closing the gap in our theoretical understanding of this model. As observed in prior work, and hinted at by our hardness results, the fully dynamic setting is significantly harder in continual release, with even basic graph problems requiring $\Omega(\text{poly}(n))$ additive error for dynamic streams while admitting $\tilde{O}(\text{poly}(\log(n))/\varepsilon)$ additive error in the insertion-only case. In this context, it would be especially interesting to deepen our understanding of the interplay between the dynamicity of the continual release setting (insertion-only vs fully-dynamic) and the *space* lower bounds (as opposed to error lower bounds) imposed by privacy. This is an area that has only recently received attention (Dinur et al., 2023) and is an intriguing future direction to explore.

Acknowledgments

Felix Zhou is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Postgraduate Scholarship (PGS D) 587420–2024. Quanquan C. Liu and Felix Zhou were supported in part by the National Science Foundation (NSF) under Grant #CCF-2453323 and a Google Academic Research Award.

References

K. J. Ahn and S. Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In L. Aceto, M. Henzinger, and J. Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 526–538, Berlin, Heidelberg, 2011. Springer. ISBN 9783642220111. https://doi.org/10.1007/978-3-642-22012-8_42.

N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29, 1996. https://doi.org/10.1145/237814.237823.

R. Arora and J. Upadhyay. On differentially private graph sparsification and applications. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13378–13389, 2019. https://proceedings.neurips.cc/paper/2019/hash/e44e875c12109e4fa3716c05008048b2-Abstract.html.

S. Assadi. A two-pass (conditional) lower bound for semi-streaming maximum matching. In J. S. Naor and N. Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 708–742. SIAM, 2022. https://doi.org/10.1137/1.9781611977073.32.

S. Assadi, M. Bateni, A. Bernstein, V. S. Mirrokni, and C. Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1616–1635. SIAM, 2019. https://doi.org/10.1137/1.9781611975482.98.

S. Assadi, S. C. Liu, and R. E. Tarjan. *An Auction Algorithm for Bipartite Matching in Streaming and Massively Parallel Computation Models*, pages 165–171. 2021. https://doi.org/10.1137/1.9781611976496.18.

S. Assadi, A. Jambulapati, Y. Jin, A. Sidford, and K. Tian. Semi-streaming bipartite matching in fewer passes and optimal space. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 627–669. SIAM, 2022. https://doi.org/10.1137/1.9781611977073.29.

B. Balle, G. Barthe, and M. Gaboardi. Privacy amplification by subsampling: Tight analyses via couplings and divergences. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6280–6290, 2018. https://proceedings.neurips.cc/paper/2018/hash/3b5020bb891119b9f5130f1fea9bd773-Abstract.html.

P. Berenbrink, B. Krayenhoff, and F. Mallmann-Trenn. Estimating the number of connected components in sublinear time. *Inf. Process. Lett.*, 114(11):639–642, 2014. https://doi.org/10.1016/j.ipl.2014.05.008.

S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In R. A. Servedio and R. Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*,

pages 173–182. ACM, 2015. https://doi.org/10.1145/2746539.2746592.

J. Blocki, E. Grigorescu, and T. Mukherjee. Privately estimating graph parameters in sublinear time. In M. Bojanczyk, E. Merelli, and D. P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, Paris, France, July 4-8, 2022*, volume 229 of *LIPIcs*, pages 26:1–26:19. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. https://doi.org/10.4230/LIPIcs.ICALP.2022.26.

J. Blocki, E. Grigorescu, T. Mukherjee, and S. Zhou. How to make your approximation algorithm private: A black-box differentially-private transformation for tunable approximation algorithms of functions with low sensitivity. In N. Megow and A. D. Smith, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2023, September 11-13, 2023, Atlanta, Georgia, USA*, volume 275 of *LIPIcs*, pages 59:1–59:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. https://doi.org/10.4230/LIPICS.APPROX/RANDOM.2023.59.

D. Boob, Y. Gao, R. Peng, S. Sawlani, C. Tsourakakis, D. Wang, and J. Wang. Flowless: Extracting densest subgraphs without flow computations. In *Proceedings of The Web Conference 2020*, pages 573–583, 2020. https://doi.org/10.1145/3366423.3380140.

M. Bun, M. Eliás, and J. Kulkarni. Differentially private correlation clustering. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1136–1146. PMLR, PMLR, 2021. http://proceedings.mlr.press/v139/bun21a.html.

T. H. Chan, M. Li, E. Shi, and W. Xu. Differentially private continual monitoring of heavy hitters from distributed streams. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 140–159, 2012. https://doi.org/10.1007/978-3-642-31680-7_8.

T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3), nov 2011. ISSN 1094-9224. https://doi.org/10.1145/2043621.2043626.

C. Chekuri, K. Quanrud, and M. R. Torres. Densest subgraph: Supermodularity, iterative peeling, and flow. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1531–1555. SIAM, 2022. https://doi.org/10.1137/1.9781611977073.64.

H. Chen, V. Cohen-Addad, T. d'Orsi, A. Epasto, J. Imola, D. Steurer, and S. Tiegel. Private estimation algorithms for stochastic block models and mixture models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, volume 36, pages 68134–68183, 2023a. http://papers.nips.cc/paper_files/paper/2023/hash/d702d78b2468d2bc80b22a2fc3e59faf-Abstract-Conference.html.

X. Chen, R. Chitnis, P. Eades, and A. Wirth. Sublinear-space streaming algorithms for estimating graph parameters on sparse graphs. In *Algorithms and Data Structures Symposium*, pages 247–261. Springer, 2023b. https://doi.org/10.1007/978-3-031-38906-1_17.

G. Cormode, H. Jowhari, M. Monemizadeh, and S. Muthukrishnan. The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. In K. Pruhs and C. Sohler, editors, *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, volume 87 of *LIPIcs*, pages 29:1–29:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. https://doi.org/10.4230/LIPICS.ESA.2017.

29.

G. Cormode, J. Dark, and C. Konrad. Approximating the caro-wei bound for independent sets in graph streams. In *International Symposium on Combinatorial Optimization*, pages 101–114. Springer, 2018. https://doi.org/10.1007/978-3-319-96151-4_9.

A. De. Lower bounds in differential privacy. In R. Cramer, editor, *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*, pages 321–338. Springer, 2012. https://doi.org/10.1007/978-3-642-28914-9_18.

L. Dhulipala, Q. C. Liu, S. Raskhodnikova, J. Shi, J. Shun, and S. Yu. 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*, pages 754–765. IEEE, 2022. https://doi.org/10.1109/FOCS54457.2022.00077.

L. Dhulipala, G. Z. Li, and Q. C. Liu. Near-optimal differentially private k-core decomposition. *CoRR*, abs/2312.07706, 2023. https://doi.org/10.48550/arXiv.2312.07706.

M. Dinitz, S. Kale, S. Lattanzi, and S. Vassilvitskii. Almost tight bounds for differentially private densest subgraph. In Y. Azar and D. Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2908–2950. SIAM, 2025a. https://doi.org/10.1137/1.9781611978322.94.

M. Dinitz, G. Z. Li, Q. C. Liu, and F. Zhou. Differentially private matchings. *CoRR*, abs/2501.00926, 2025b. https://doi.org/10.48550/arXiv.2501.00926.

Y. Dinitz. Dinitz' algorithm: The original version and Even's version. In *Theoretical Computer Science: Essays in Memory of Shimon Even*, pages 218–240. Springer, 2006. https://doi.org/10.1007/11685654_10.

I. Dinur and K. Nissim. Revealing information while preserving privacy. In F. Neven, C. Beeri, and T. Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 202–210. ACM, 2003. https://doi.org/10.1145/773153.773173.

I. Dinur, U. Stemmer, D. P. Woodruff, and S. Zhou. On differential privacy and adaptive data analysis with bounded space. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 35–65. Springer, 2023. https://doi.org/10.1007/978-3-031-30620-4_2.

M. Dupré la Tour, M. Henzinger, and D. Saulpic. Differential privacy for clustering under continual observation. *CoRR*, abs/2307.03430, 2023. https://doi.org/10.48550/arXiv.2307.03430.

C. Dwork and J. Lei. Differential privacy and robust statistics. In M. Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 371–380. ACM, 2009. https://doi.org/10.1145/1536414.1536466.

C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. https://doi.org/10.1561/0400000042.

C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In S. Halevi and T. Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-32732-5. https://doi.org/10.1007/11681878_14.

C. Dwork, F. McSherry, and K. Talwar. The price of privacy and the limits of LP decoding. In D. S. Johnson and U. Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 85–94. ACM, 2007. https://doi.org/10.1145/1250790.1250804.

C. Dwork, M. Naor, O. Reingold, G. N. Rothblum, and S. Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 381–390, 2009. https://doi.org/10.1145/1536414.1536467.

C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In L. J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 715–724. ACM, 2010a. https://doi.org/10.1145/1806689.1806787.

C. Dwork, G. N. Rothblum, and S. P. Vadhan. Boosting and differential privacy. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, Las Vegas, Nevada, USA, October 23-26, 2010*, pages 51–60. IEEE Computer Society, 2010b. https://doi.org/10.1109/FOCS.2010.12.

T. Eden, Q. C. Liu, S. Raskhodnikova, and A. D. Smith. Triangle counting with local edge differential privacy. In K. Etessami, U. Feige, and G. Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 52:1–52:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. https://doi.org/10.4230/LIPIcs.ICALP.2023.52.

J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965. https://doi.org/10.6028/jres.069B.013.

A. Epasto, J. Mao, A. M. Medina, V. Mirrokni, S. Vassilvitskii, and P. Zhong. Differentially private continual releases of streaming frequency moment estimations. In Y. T. Kalai, editor, *14th Innovations in Theoretical Computer Science Conference, ITCS 2023, January 10-13, 2023, MIT, Cambridge, Massachusetts, USA*, volume 251 of *LIPIcs*, pages 48:1–48:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. https://doi.org/10.4230/LIPIcs.ITCS.2023.48.

H. Esfandiari, M. Hajiaghayi, and D. P. Woodruff. Brief announcement: Applications of uniform sampling: Densest subgraph and beyond. In C. Scheideler and S. Gilbert, editors, *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 397–399. ACM, 2016. https://doi.org/10.1145/2935764.2935813.

H. Esfandiari, S. Lattanzi, and V. S. Mirrokni. Parallel and streaming algorithms for k-core decomposition. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1396–1405. PMLR, 2018. http://proceedings.mlr.press/v80/esfandiari18a.html.

A. Farhadi, M. Hajiaghayi, and E. Shi. Differentially private densest subgraph. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, *International Conference on Artificial Intelligence and Statistics, AISTATS 2022, 28-30 March 2022, Virtual Event*, volume 151 of *Proceedings of Machine Learning Research*, pages 11581–11597. PMLR, 2022. https://proceedings.mlr.press/v151/farhadi22a.html.

J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005. https:

//doi.org/10.1016/j.tcs.2005.09.013.

H. Fichtenberger, M. Henzinger, and W. Ost. Differentially private algorithms for graphs under continual observation. In P. Mutzel, R. Pagh, and G. Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 42:1–42:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. https://doi.org/10.4230/LIPIcs.ESA.2021.42.

H. Fichtenberger, M. Henzinger, and J. Upadhyay. Constant matters: Fine-grained error bound on differentially private continual observation. In *International Conference on Machine Learning*, pages 10072–10092. PMLR, 2023. https://proceedings.mlr.press/v202/fichtenberger23a.html.

M. Fischer, S. Mitrovic, and J. Uitto. Deterministic $(1+\varepsilon)$-approximate maximum matching with poly$(1/\varepsilon)$ passes in the semi-streaming model and beyond. In S. Leonardi and A. Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, STOC 2022, pages 248–260, New York, NY, USA, 2022. ACM. ISBN 9781450392648. https://doi.org/10.1145/3519935.3520039.

P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985. https://doi.org/10.1016/0022-0000(85)90041-8.

P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics & Theoretical Computer Science*, (Proceedings), 2007. https://doi.org/10.46298/dmtcs.3545.

P. Ghosh and M. Stoeckl. Low-memory algorithms for online edge coloring. In K. Bringmann, M. Grohe, G. Puppis, and O. Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 71:1–71:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. https://doi.org/10.4230/LIPICS.ICALP.2024.71.

B. V. Halldórsson, M. M. Halldórsson, E. Losievskaja, and M. Szegedy. Streaming algorithms for independent sets in sparse hypergraphs. *Algorithmica*, 76:490–501, 2016. https://doi.org/10.1007/s00453-015-0051-5.

M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 61–70, 2010. https://doi.org/10.1109/FOCS.2010.85.

M. Henzinger, A. R. Sricharan, and T. A. Steiner. Differentially private histogram, predecessor, and set cardinality under continual observation. *CoRR*, abs/2306.10428, 2023. https://doi.org/10.48550/arXiv.2306.10428.

M. Henzinger, A. R. Sricharan, and L. Zhu. Tighter bounds for local differentially private core decomposition and densest subgraph. *CoRR*, abs/2402.18020, 2024a. https://doi.org/10.48550/arXiv.2402.18020.

M. Henzinger, J. Upadhyay, and S. Upadhyay. A unifying framework for differentially private sums under continual observation. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 995–1018. SIAM, 2024b. https://doi.org/10.1137/1.9781611977912.38.

M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998. https://doi.org/10.1090/dimacs/050/05.

Z. Huang and P. Peng. Dynamic graph stream algorithms in o(n) space. *Algorithmica*, 81 (5):1965–1987, 2019. https://doi.org/10.1007/S00453-018-0520-8.

J. Imola, A. Epasto, M. Mahdian, V. Cohen-Addad, and V. Mirrokni. Differentially private hierarchical clustering with provable approximation guarantees. In *International Conference on Machine Learning*, pages 14353–14375. PMLR, 2023. https://proceedings.mlr.press/v202/imola23a.html.

P. Jain, I. Kalemaj, S. Raskhodnikova, S. Sivakumar, and A. D. Smith. Counting distinct elements in the turnstile model with differential privacy under continual observation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023a. http://papers.nips.cc/paper_files/paper/2023/hash/0ef1afa0daa888d695dcd5e9513bafa3-Abstract-Conference.html.

P. Jain, S. Raskhodnikova, S. Sivakumar, and A. D. Smith. The price of differential privacy under continual observation. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 14654–14678. PMLR, 2023b. https://proceedings.mlr.press/v202/jain23b.html.

P. Jain, A. D. Smith, and C. Wagaman. Time-aware projections: Truly node-private graph statistics under continual observation. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 127–145. IEEE, 2024. https://doi.org/10.1109/SP54263.2024.00196.

I. Kalemaj, S. Raskhodnikova, A. Smith, and C. E. Tsourakakis. Node-differentially private estimation of the number of connected components. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 183–194, 2023. https://doi.org/10.1145/3584372.3588671.

S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 457–476. Springer, 2013. https://doi.org/10.1007/978-3-642-36594-2_26.

V. King, A. Thomo, and Q. Yong. Computing (1+epsilon)-approximate degeneracy in sublinear time. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 2160–2168. ijcai.org, 2023. https://doi.org/10.24963/IJCAI.2023/240.

J. Liu, J. Upadhyay, and Z. Zou. Optimal bounds on private graph approximation. In D. P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 1019–1049. SIAM, 2024. https://doi.org/10.1137/1.9781611977912.39.

Q. C. Liu, J. Shi, S. Yu, L. Dhulipala, and J. Shun. Parallel batch-dynamic algorithms for k-core decomposition and related graph problems. In *34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 191–204, 2022. https://doi.org/10.1145/3490148.3538569.

M. Lyu, D. Su, and N. Li. Understanding the sparse vector technique for differential privacy. *Proceedings of the VLDB Endowment*, 10(6):637–648, 2017. https://doi.org/10.14778/3055330.3055331.

D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983. https://doi.org/10.1145/2402.322385.

A. McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014. https://doi.org/10.1145/2627692.2627694.

A. McGregor and S. Vorotnikova. A simple, space-efficient, streaming algorithm for matchings in low arboricity graphs. In R. Seidel, editor, *1st Symposium on Simplicity in Algorithms, SOSA 2018, New Orleans, LA, USA, January 7-10, 2018*, volume 61 of *OASIcs*, pages 14:1–14:4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. https://doi.org/10.4230/OASIcs.SOSA.2018.14.

A. McGregor, D. Tench, S. Vorotnikova, and H. T. Vu. Densest subgraph in dynamic graph streams. In *International Symposium on Mathematical Foundations of Computer Science*, pages 472–482. Springer, 2015. https://doi.org/10.1007/978-3-662-48054-0_39.

D. J. Mir, S. Muthukrishnan, A. Nikolov, and R. N. Wright. Pan-private algorithms via statistics on sketches. In M. Lenzerini and T. Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 37–48. ACM, 2011. https://doi.org/10.1145/1989284.1989290.

M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. ISBN 978-0-521-83540-4. https://doi.org/10.1017/CBO9780511813603.

R. Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978. https://doi.org/10.1145/359619.359627.

T. T. Mueller, D. Usynin, J. C. Paetzold, R. Braren, D. Rueckert, and G. Kaissis. Differentially private guarantees for analytics and machine learning on graphs: A survey of results. *Journal of Privacy and Confidentiality*, 14(1), 2024. https://doi.org/10.29012/jpc.820.

S. Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005. https://doi.org/10.1561/0400000002.

D. Nguyen and A. Vullikanti. Differentially private densest subgraph detection. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8140–8151. PMLR, 2021. http://proceedings.mlr.press/v139/nguyen21i.html.

K. Nissim, S. Raskhodnikova, and A. D. Smith. Smooth sensitivity and sampling in private data analysis. In D. S. Johnson and U. Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 75–84. ACM, 2007. https://doi.org/10.1145/1250790.1250803.

S. Raskhodnikova and A. D. Smith. Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 495–504, 2016a. https://doi.org/10.1109/FOCS.2016.60.

S. Raskhodnikova and A. D. Smith. Differentially private analysis of graphs. In *Encyclopedia of Algorithms*, pages 543–547. 2016b. https://doi.org/10.1007/978-1-4939-2864-4_549.

S. Raskhodnikova and T. A. Steiner. Fully dynamic algorithms for graph databases with edge differential privacy. *Proc. ACM Manag. Data*, 3(2):99:1–99:28, 2025. https://doi.org/10.1145/3725236.

A. Roth and T. Roughgarden. Interactive privacy via the median mechanism. In L. J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 765–774. ACM, 2010. https://doi.org/10.1145/1806689.1806794.

A. E. Sariyüce, B. Gedik, G. Jacques-Silva, K. Wu, and Ü. V. Çatalyürek. Streaming algorithms for k-core decomposition. *Proc. VLDB Endow.*, 6(6):433–444, apr 2013. ISSN 2150-8097. https://doi.org/10.14778/2536336.2536344.

S. Song, S. Little, S. Mehta, S. A. Vinterbo, and K. Chaudhuri. Differentially private continual release of graph statistics. *CoRR*, abs/1809.02575, 2018. https://doi.org/10.48550/arXiv.1809.02575.

B. Sun, T.-H. H. Chan, and M. Sozio. Fully dynamic approximate k-core decomposition in hypergraphs. *ACM Trans. Knowl. Discov. Data*, 14(4), may 2020. https://doi.org/10.1145/3385416.

J. Upadhyay. Random projections, graph sparsification, and differential privacy. In K. Sako and P. Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 276–295. Springer, 2013. https://doi.org/10.1007/978-3-642-42033-7_15.

J. Upadhyay. Sublinear space private algorithms under the sliding window model. In *International Conference on Machine Learning*, pages 6363–6372. PMLR, 2019. http://proceedings.mlr.press/v97/upadhyay19a.html.

J. Upadhyay, S. Upadhyay, and R. Arora. Differentially private analysis on graph streams. In *International Conference on Artificial Intelligence and Statistics*, pages 1171–1179. PMLR, 2021. http://proceedings.mlr.press/v130/upadhyay21a.html.

F. Zhou. Continual release of densest subgraphs: Privacy amplification & sublinear space via subsampling. In S. Assadi and E. Rotenberg, editors, *2026 Symposium on Simplicity in Algorithms, SOSA 2026, Vancouver, BC, Canada, January 12-14, 2026*, pages 170–191. SIAM, 2026. https://doi.org/10.1137/1.9781611978964.13.

Appendix A. Deferred Preliminaries

We now remind the reader of some standard definitions and tools.

A.1. **Approximation Algorithms.** We consider optimization problems in both the minimization and maximization setting. Let $\mathrm{OPT} \in \mathbb{R}$ be the optimum value for the problem. For a minimization problem, a (β, ζ)-*approximate algorithm* outputs a solution with cost at most $\beta \cdot \mathrm{OPT} + \zeta$. For a maximization problem, a (β, ζ)-approximate algorithm outputs a solution of value at least $\frac{1}{\beta} \cdot \mathrm{OPT} - \zeta$. For estimation problems, a (β, ζ)-approximate algorithm outputs an estimate $\tilde{R} \in \mathbb{R}$ of some quantity $R \in \mathbb{R}$ where $R - \zeta \leq \tilde{R} \leq \beta \cdot R + \zeta$ (one-sided multiplicative error). As a shorthand, we write a β-approximation algorithm to indicate a $(\beta, 0)$-approximation algorithm. Our approximation bounds hold, with high probability, over all releases.

A.2. **Concentration Inequalities.** Below, we state the version of the multiplicative Chernoff bound we use throughout this work.

Theorem A.1 (Multiplicative Chernoff Bound; Theorems 4.4, 4.5 in (Mitzenmacher and Upfal, 2005))**.** *Let $X = \sum_{i=1}^{n} X_i$ where each X_i is a Bernoulli variable which takes value 1 with probability p_i and value 0 with probability $1 - p_i$. Let $\mu = \mathbb{E}[X] = \sum_{i=1}^{n} p_i$. Then, it holds:*

(1) *Upper Tail:* $\mathbf{P}[X \geq (1+\psi) \cdot \mu] \leq \exp\left(-\frac{\psi^2 \mu}{2+\psi}\right)$ *for all* $\psi > 0$*;*

(2) *Lower Tail:* $\mathbf{P}[X \leq (1-\psi) \cdot \mu] \leq \exp\left(-\frac{\psi^2 \mu}{3}\right)$ *for all* $0 < \psi < 1$*.*

A.3. **Differential Privacy Tools.** Here, we define the privacy tools commonly used in differential privacy in terms of the continual release model. Throughout the paper, we use some standard privacy mechanisms as building blocks (see (Dwork and Roth, 2014) for a reference).

Definition A.2 (Global sensitivity)**.** The global sensitivity of a function $f : \mathcal{D} \to \mathbb{R}^d$ is defined by

$$\Delta_f = \max_{D, D' \in \mathcal{D}, D \sim D'} \|f(D) - f(D')\|_1.$$

where $D \sim D'$ are neighboring datasets and differ by an element.

Definition A.3 (Laplace Distribution)**.** We say a random variable X is drawn from a Laplace distribution with mean μ and scale $b > 0$ if the probability density function of X at x is $\frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$. We use the notation $X \sim \mathsf{Lap}(b)$ to denote that X is drawn from the Laplace distribution with scale b and mean $\mu = 0$.

The Laplace mechanism for $f : X \to \mathbb{R}$ with global sensitivity σ adds Laplace noise to the output of f with scale $b = \sigma/\varepsilon$ before releasing.

Proposition A.4 (Theorem 3.6 in (Dwork and Roth, 2014))**.** *The Laplace mechanism is ε-DP.*

Theorem A.5 (Adaptive Composition; (Dwork et al., 2006; Dwork and Lei, 2009; Dwork et al., 2010b))**.** *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).*

Theorem A.6 (Group Privacy; Theorem 2.2 in (Dwork and Roth, 2014))**.** *Given an* ε*-edge (node) differentially private algorithm,* $\mathcal{A}$*, for all pairs of input streams* S *and* S'*, it holds that for all possible outcomes* $Y \in \text{Range}(\mathcal{A})$*,*

$$e^{-k\varepsilon} \leq \frac{\mathbf{P}[\mathcal{A}(S') \in Y]}{\mathbf{P}[\mathcal{A}(S) \in Y]} \leq e^{k\varepsilon}$$

where k *is the edge (node) edit distance between* S *and* S'*.*

A.3.1. *Sparse Vector Technique.* Below, we define the *sparse vector technique* and give its privacy and approximation guarantees. The sparse vector technique is used to answer *above threshold* queries where an above threshold query checks whether the output of a function that operates on an input graph G exceeds a threshold T.

We use the variant introduced by Lyu et al. (2017) and used by Fichtenberger et al. (2021). Let D be an arbitrary (graph) dataset, (f_t, τ_t) a sequence of (possibly adaptive) query-threshold pairs, Δ an upper bound on the maximum sensitivity of all queries f_t, and an upper bound c on the maximum number of queries to be answered "above". Typically, the AboveThreshold algorithm stops running at the first instance of the input exceeding the threshold, but we use the variant where the input can exceed the threshold at most c times where c is a parameter passed into the function.

Throughout this paper, we use the class $\text{SVT}(\varepsilon, \Delta, c)$ (Algorithm A.1) where ε is our privacy parameter, Δ is an upper bound on the maximum sensitivity of incoming queries, and c is the maximum number of "above" queries we can make. The class provides a ProcessQuery($query, threshold$) function where *query* is the query to SVT and *threshold* is the threshold that we wish to check whether the query exceeds.

Theorem A.7 (Theorem 2 in (Lyu et al., 2017))**.** *Algorithm A.1 is* ε*-DP.*

We remark that the version of SVT we employ (Algorithm A.1) does not require us to resample the noise for the thresholds (Algorithm A.1, Line 4) after each query but we do need to resample the noise (Algorithm A.1, Line 9) for the queries after each query.

Appendix B. Non-Private Streaming k-Core Decomposition

In this section, we convert our private k-core decomposition algorithm to a non-private k-core decomposition algorithm. The algorithm keeps the same sampled level-data-structure framework, but removes all privacy-specific mechanisms: there is no privacy parameter, no per-vertex sparse-vector instance, and no noisy threshold test to determine whether a node moves up a level. Instead, every promotion decision is made by a deterministic comparison against the sampled degree, and the sampling probabilities are chosen so that a Chernoff concentration bound alone suffices.

Algorithm A.1: Sparse Vector Technique

```
Input: privacy budget ε, upper bound on query sensitivity Δ, maximum allowed
  "above" answers c
Class SVT(ε, Δ, c)
    ε_1, ε_2 ← ε/2
    ρ ← Lap(Δ/ε_1)
    count ← 0
    Function ProcessQuery(f_t(D), τ_t)
        if count > c then
            return "abort"
        if f_t(D) + Lap(2cΔ/ε_2) ≥ τ_t + ρ then
            return "above"
            count ← count +1
        else
            return "below"
```

Our algorithm is still distinct from the streaming algorithm of Esfandiari et al. (2018) because it uses the level-data-structure viewpoint of Liu et al. (2022) together with an edge-sampling scheme that depends on the level partitions of the nodes.

B.1. **Algorithm Description.** For each scale j, we maintain a sampled subgraph X_j whose sampling rate is tuned to the degree scale $(1+\eta)^j$. The key point is that the promotion threshold in group j is

$$\tau_j = p_j(1+\eta)^{j-1},$$

while the two relevant full-graph degree scales are $(1+\eta)^{j-2}$ and $(1+\eta)^j$. Thus there is a multiplicative gap of a factor of $(1+\eta)$ on either side of the threshold. This gap is enough to absorb the sampling error via Chernoff bounds, so no additive term is needed in the non-private analysis.

For every group j and every vertex v, we store a current level $levels[j][v] \in \{0,\dots,F-1\}$. After each stream update, the algorithm scans the levels bottom-up. If the sampled upper-degree of v in group j at its current level is at least τ_j, then v is promoted by one level. As in the private algorithm, the largest group whose top level contains v determines the released estimate for v.

In the analysis below, let $\mathcal{K}_t$ denote the hypothetical family of level data structures at time t that keeps *all* edges of the graph, but places every vertex at exactly the same level as in the sampled data structures maintained by Algorithm B.1. For $\ell \in \{0,\dots,F-1\}$, $j \in Q$, and $v \in V$, let $\deg_{\mathcal{K}_t}(\ell, j, v)$ be the induced degree of v in the j-th data structure of $\mathcal{K}_t$ restricted to neighbors on level at least ℓ.

Also let $Y_t(\ell, j, v)$ denote the sampled counterpart of this quantity, i.e., the number of sampled neighbors of v in X_j that lie on level at least ℓ in group j at time t. Conditional on the current graph and the current level assignment, $Y_t(\ell, j, v)$ is a sum of independent Bernoulli random variables, because each relevant edge was sampled independently when it arrived.

Algorithm B.1: Data structure for non-private k-core decomposition for adaptive insertion-only streams

Class `SampledCore`(η, n, T)
  $J \leftarrow \lceil 2\log_{(1+\eta)}(n) \rceil$
  $Q \leftarrow \{0, \dots, J\}$
  $F \leftarrow J + 1$ ($F-1$ denotes the topmost level of any level data structure)
  Maintain sampled edges $X_j \leftarrow \varnothing$ for each $j \in Q$
  for $j \in Q$ **do**
    Initialize
$$p_j \leftarrow \min\left\{1, \frac{c\log(nT)}{\eta^2(1+\eta)^j}\right\},$$
    where $c > 0$ is a sufficiently large fixed constant
    for $v \in V$ **do**
      Initialize $levels[j][v] \leftarrow 0$
  Function `SampleEdge`$(e_t = \{u, v\})$
    for $j \in Q$ **do**
      if $levels[j][u] < F-1$ ***or*** $levels[j][v] < F-1$ **then**
        Add e_t to X_j with probability p_j
  Function `UpdateLevels()`
    for *level* $\ell \in \{0, \dots, F-2\}$ **do**
      for $j \in Q$ **do**
        for $v \in V$ **do**
          if $levels[j][v] \neq \ell$ **then**
            continue
          if $\deg^+_{X_j}(v) \geq p_j(1+\eta)^{j-1}$ **then**
            $levels[j][v] \leftarrow levels[j][v] + 1$
  Function `GetLevels()`
    return *levels*

Lemma B.1. *There is an event $\mathcal{E}$ that holds with probability at least $1-(nT)^{-10}$ such that, simultaneously for every update time $t \in [T]$, every $j \in Q$, every level $\ell \in \{0, \dots, F-1\}$, and every vertex $v \in V$, the following two statements hold:*
(1) *if $\deg_{\mathcal{K}_t}(\ell, j, v) \geq (1+\eta)^j$, then $Y_t(\ell, j, v) \geq p_j(1+\eta)^{j-1}$;*
(2) *if $\deg_{\mathcal{K}_t}(\ell, j, v) \leq (1+\eta)^{j-2}$, then $Y_t(\ell, j, v) < p_j(1+\eta)^{j-1}$.*

Proof. Fix t, j, ℓ, v. If $p_j = 1$, then $Y_t(\ell, j, v) = \deg_{\mathcal{K}_t}(\ell, j, v)$ exactly, so both claims immediately follow. Hence suppose $p_j < 1$.

For the first claim, let
$$d = \deg_{\mathcal{K}_t}(\ell, j, v) \qquad \text{and} \qquad \mu = \mathbb{E}[Y_t(\ell, j, v)] = p_j d.$$

Algorithm B.2: Non-private k-core decomposition for adaptive insertion-only streams

```
Initialize SampledCore(η, n, T)
levels ← SampledCore.GetLevels()
for each new update e_t do
    if e_t ≠ ⊥ then
        SampledCore.SampleEdge(e_t)
    SampledCore.UpdateLevels()
    levels ← SampledCore.GetLevels()
    for every vertex v ∈ V do
        if {j ∈ Q : levels[j][v] = F − 1} = ∅ then
            Release v's core estimate as 0
        else
            j_top(v) ← max{j ∈ Q : levels[j][v] = F − 1}
            Release v's core estimate as (2 + η) · (1 + η)^{j_top(v)}
```

If $d \geq (1+\eta)^j$, then

$$\mu \geq p_j(1+\eta)^j \geq \frac{c\log(nT)}{\eta^2}.$$

Moreover, the promotion threshold is

$$p_j(1+\eta)^{j-1} \leq \frac{\mu}{1+\eta}.$$

Set

$$\delta = 1 - \frac{1}{1+\eta} = \frac{\eta}{1+\eta}.$$

Then by the multiplicative Chernoff bound,

$$\Pr\left[Y_t(\ell,j,v) < p_j(1+\eta)^{j-1}\right] \leq \Pr[Y_t(\ell,j,v) \leq (1-\delta)\mu] \leq \exp\left(-\frac{\delta^2\mu}{2}\right) \leq \exp\left(-\frac{c\log(nT)}{8}\right),$$

where the last inequality uses $\eta \in (0,1)$.

For the second claim, the probability of crossing a fixed threshold is maximized when the mean is as large as possible, so it suffices to consider the worst case $d = (1+\eta)^{j-2}$. In that case

$$\mu^\star = p_j(1+\eta)^{j-2} = \frac{p_j(1+\eta)^{j-1}}{1+\eta} \geq \frac{c\log(nT)}{\eta^2(1+\eta)^2}.$$

Therefore,

$$\Pr\left[Y_t(\ell,j,v) \geq p_j(1+\eta)^{j-1}\right] \leq \Pr[Y_t(\ell,j,v) \geq (1+\eta)\mu^\star] \leq \exp\left(-\frac{\eta^2\mu^\star}{3}\right) \leq \exp\left(-\frac{c\log(nT)}{12}\right).$$

There are at most $nT|Q|F$ tuples (t,j,ℓ,v) to consider, and $|Q|, F = O(\log_{1+\eta} n) = O(\eta^{-1}\log n)$. Choosing the constant c large enough and taking a union bound proves the lemma. □

B.2. **Approximation Guarantee.** We now prove the two structural invariants needed in the level-data-structure analysis. Unlike in the private proof, there is no additive slack coming from noisy threshold tests.

Lemma B.2. *If vertex v is in level $\ell < F-1$ of group j after a call to* UpdateLevels*, then conditioned on the event $\mathcal{E}$ from Theorem B.1,*

$$\deg_{\mathcal{K}_t}(\ell, j, v) < (1+\eta)^j.$$

Consequently, with high probability,

$$\deg_{\mathcal{K}_t}(\ell, j, v) \le (1+\eta)^j.$$

Proof. Condition on the event $\mathcal{E}$. Since v is still in level ℓ after the call to UpdateLevels, the algorithm did *not* promote v when it processed level ℓ in group j. Therefore

$$Y_t(\ell, j, v) < p_j(1+\eta)^{j-1}.$$

By Theorem B.1, this is impossible if $\deg_{\mathcal{K}_t}(\ell, j, v) \ge (1+\eta)^j$. Hence $\deg_{\mathcal{K}_t}(\ell, j, v) < (1+\eta)^j$, as claimed. □

Lemma B.3. *If vertex v is in level $\ell > 0$ of group j after a call to* UpdateLevels*, then conditioned on the event $\mathcal{E}$ from Theorem B.1,*

$$\deg_{\mathcal{K}_t}(\ell-1, j, v) > (1+\eta)^{j-2}.$$

Consequently, with high probability,

$$\deg_{\mathcal{K}_t}(\ell-1, j, v) \ge (1+\eta)^{j-2}.$$

Proof. Condition on the event $\mathcal{E}$. Since v ends in level $\ell > 0$, it must have been promoted from level $\ell-1$ to level ℓ at some point during the call to UpdateLevels. At the moment of this promotion, the algorithm observed

$$Y_t(\ell-1, j, v) \ge p_j(1+\eta)^{j-1}.$$

By Theorem B.1, this cannot happen if $\deg_{\mathcal{K}_t}(\ell-1, j, v) \le (1+\eta)^{j-2}$. Therefore

$$\deg_{\mathcal{K}_t}(\ell-1, j, v) > (1+\eta)^{j-2},$$

which proves the claim. □

For the final approximation guarantee we use the same folklore peeling fact as in the private proof.

Lemma B.4 (Folklore; (Matula and Beck, 1983))**.** *Suppose we perform the following peeling procedure. Given an input graph $G = (V, E)$, iteratively remove all vertices of degree at most $d^\star$ until no such vertex remains. Then every removed vertex v satisfies $k(v) \le d^\star$, while every vertex w that remains at the end satisfies $k(w) > d^\star$.*

Lemma B.5. *Algorithm B.2 returns a $(2+\eta)(1+\eta)^5$-approximate k-core decomposition with high probability. Equivalently, for every vertex v,*

$$k(v) \le \hat{k}(v) \le (2+\eta)(1+\eta)^5 k(v),$$

where $\hat{k}(v)$ is the released estimate.

Proof. Condition on the event $\mathcal{E}$ from Theorem B.1. Under this event, Lemma B.2 and Lemma B.3 give exact multiplicative versions of the two invariants used in the proof of Theorem 4.7 of Dhulipala et al. (2022). Thus the same peeling/pruning argument goes through, but now with no additive term.

We first record the upper implication produced by the peeling part of the argument: for every integer $g \geq 0$,

$$\hat{k}(i) \leq (2+\eta)(1+\eta)^g \quad \Longrightarrow \quad k(i) \leq (1+\eta)^{g+1}. \tag{B.1}$$

Indeed, once a vertex fails to reach the top level in the next group, the vertices in the lower levels can be peeled using Lemma B.2, and Lemma B.4 then implies the above bound exactly as in the private proof, but without any additive slack.

Next, the pruning part of the same argument shows that for every integer $g' \geq 0$,

$$\hat{k}(i) \geq (1+\eta)^{g'} \quad \Longrightarrow \quad k(i) \geq \frac{(1+\eta)^{g'-2}}{(2+\eta)(1+\eta)^2}. \tag{B.2}$$

This is the same lower-bound statement as in the private proof, except that the additive term disappears. The only loss relative to the standard LDS proof is the same shift by two powers of $(1+\eta)$, which contributes the extra multiplicative factor $(1+\eta)^2$.

Now fix a vertex i and suppose

$$\hat{k}(i) = (2+\eta)(1+\eta)^g$$

for some $g \geq 0$.

Applying (B.1) with this value of g gives

$$k(i) \leq (1+\eta)^{g+1} < (2+\eta)(1+\eta)^g = \hat{k}(i),$$

and hence $k(i) \leq \hat{k}(i)$.

For the reverse inequality, let

$$g' \triangleq g + \lfloor \log_{(1+\eta)}(2+\eta) \rfloor.$$

Then

$$g' \geq g + \log_{(1+\eta)}(2+\eta) - 1$$

and therefore

$$\hat{k}(i) = (2+\eta)(1+\eta)^g \geq (1+\eta)^{g'}.$$

Applying (B.2), we obtain

$$k(i) \geq \frac{(1+\eta)^{g'-2}}{(2+\eta)(1+\eta)^2} \geq \frac{(1+\eta)^{g+\log_{(1+\eta)}(2+\eta)-3}}{(2+\eta)(1+\eta)^2}.$$

Since

$$(1+\eta)^{g+\log_{(1+\eta)}(2+\eta)-3} = \frac{(2+\eta)(1+\eta)^g}{(1+\eta)^3} = \frac{\hat{k}(i)}{(1+\eta)^3},$$

we conclude that

$$k(i) \geq \frac{\hat{k}(i)}{(2+\eta)(1+\eta)^5},$$

or equivalently,

$$\hat{k}(i) \leq (2+\eta)(1+\eta)^5 k(i).$$

Combining the two bounds proves that

$$k(i) \leq \hat{k}(i) \leq (2+\eta)(1+\eta)^5 k(i)$$

for every vertex i. Since $\Pr[\mathcal{E}] \geq 1 - (nT)^{-10}$, the guarantee holds with high probability. □

Remark. Because

$$(2+\eta)(1+\eta)^5 = 2 + O(\eta),$$

one can equivalently state the guarantee as a $(2+\gamma)$-approximation for any fixed constant $\gamma > 0$ by running the algorithm with a sufficiently smaller internal slack parameter $\eta = \Theta(\gamma)$.

Appendix C. Proof of Theorem 5.1

In this section, we restate and prove Theorem 5.1.

Theorem C.1 (Sublinear Space Private k-Core Decomposition)**.** *Fix $\eta \in (0, 1]$. Algorithm 5.2 is an ε-DP algorithm for the k-core decomposition problem in the continual release model for insertion-only streams. At every $t \in [T]$, the algorithm returns a value for each vertex, such that every value is a $\left(2+\eta, O\left(\eta^{-2}\varepsilon^{-1}\log^3(n)\right)\right)$-approximation of the corresponding vertex's true core value, with probability $1 - 1/\operatorname{poly}(n)$. The maximum space used is $O\left(\eta^{-4}\varepsilon^{-1} n \log^5 n\right)$, with probability $1 - 1/\operatorname{poly}(n)$.*

Our algorithm must solve several crucial and non-trivial challenges, as mentioned in the Technical Overview (Section 2), making our algorithm conceptually more complicated than any non-private streaming k-core algorithm. We give a short reminder below of our previously detailed challenges for k-core decomposition as it is our first result. Densest subgraph and matchings share similar issues which are detailed in Section 2.

- We cannot use a differentially private k-core decomposition algorithm as a black-box because all vertices may change their core numbers at some point in the stream, resulting in $\Omega(n)$ changes from the black-box private k-core decomposition algorithm. Such a black-box usage of the private algorithm is difficult to analyze without losing a factor of $\Omega(n)$ in the pure DP setting resulting from composition.
- We cannot produce *one* uniform sample of edges from the graph since this results in an uneven distribution of edges among the cores. Suppose we sample each edge with probability p; if p is set to be too small, then vertices with smaller core numbers will not have enough adjacent edges to produce a good concentration bound.
- We must also be able to deal with vertices which have large degree and very small cores (consider a star graph). Sampling edges uniformly from the original graph, without maintaining some form of a data structure on the sampled edges, will not allow us to distinguish vertices with large degree and large core numbers from vertices with large degree but small core numbers.
- We require a composition theorem that does not lose privacy for *each* release of a new core number among the n vertices. Intuitively, we should not lose privacy for each release because for edge-neighboring insertion-only streams, only one edge differs between the two streams. This theorem uses the recently introduced multidimensional sparse vector technique (Dhulipala et al., 2023).

We give the pseudocode for our data structure and algorithm in Algorithm 5.1 and Algorithm 5.2, respectively.

C.1. **Privacy Guarantee.** We now prove the privacy guarantees of our algorithm. Specifically, we show that our algorithm maintains ε-differential privacy on edge-neighboring insertion-only streams. Note that although we are using intuition from the multidimensional sparse vector technique (MAT) given by Dhulipala et al. (2023), our proof is different in that the subsampling in our algorithm requires us to analyze the coupled sensitivity of queries (i.e., the sensitivity of the queries under a coupling of the randomness on neighboring data streams), with the coupling depending adaptively on the current output. MAT handles releasing values for n vertices without a factor-n composition loss, but was not proven to handle adaptive coupled sensitivity.

This is a subtle difference since the original mechanism works in the static setting. Hence, we call our version the *adaptive* multidimensional sparse vector technique and provide the full proof below.

Lemma C.2. *Algorithm 5.2 is ε-differentially private on edge-neighboring insertion-only streams as defined in Definition 4.2.*

We analyze the privacy guarantees using the privacy loss variable technique as employed by Zhou (2026).

Proof. Let G and G' be edge-neighboring insertion-only streams differing in exactly one edge update, $e_{t^*} = \{x, y\}$. Let R, R' denote the random coins used to subsample edges in G and G' respectively. We proceed under the coupling (R, R') where the coins for non-empty common edge updates $e_j = e'_j, j \neq t^*$ are identical.

The only algorithm state that depends on private data is the level of each node in each subgraph X_j. Under the simplification above, the levels are entirely determined by the deterministic mapping of the successes and failures of the SVT threshold queries, and we can restrict our privacy analysis to the outputs of these queries.

Given input G, let $\mathbf{B}_{t,\ell} = \{B_{t,\ell}(j,w)\}_{j\in Q, w\in V}$ denote the random variables corresponding to the outputs of the SVT queries at time t and for level ℓ. Here we interpret the SVT to output "below" if the loop over ℓ is the incorrect level (Algorithm 5.1, Line 20). Similarly, define $\mathbf{B}'_{t,\ell}$ given input G'. We define the privacy loss variable $\mathcal{L}$ over the entire stream, which evaluates the log-likelihood ratio of the transcript $\mathbf{B}_{1:T,0:F-2}$ being generated by G versus G'. Recall that an algorithm is ε-DP if and only if the absolute value of the privacy loss variable is almost surely bounded above by ε (Dwork and Roth, 2014, Lemma 3.17).

Using the chain rule, we can equivalently express $\mathcal{L}$ over time t by drawing $b_{t,\ell}(j,w) \sim B_{t,\ell}(j,w)$ for each $j \in Q$ and $w \in V$, coupled random coins $(r, r') \sim (R, R')$, and then writing

$$\begin{aligned}\mathcal{L} &:= \ln\left(\frac{\Pr[\mathbf{B}_{1:T,0:F-2} = \mathbf{b}_{1:T,0:F-2} \mid R = r]}{\Pr[\mathbf{B}'_{1:T,0:F-2} = \mathbf{b}_{1:T,0:F-2} \mid R' = r']}\right) \\ &= \sum_{t=1}^{T}\sum_{\ell=0}^{F-2}\sum_{j\in Q}\sum_{w\in V} \ln\left(\frac{\Pr[B_{t,\ell}(j,w) = b_{t,\ell}(j,w) \mid \mathbf{B}_{<t,:} = \mathbf{b}_{<t,:}, \mathbf{B}_{t,<\ell} = \mathbf{b}_{t,<\ell} \mid R = r]}{\Pr[B'_{t,\ell}(j,w) = b_{t,\ell}(j,w) \mid \mathbf{B}'_{<t,:} = \mathbf{b}'_{<t,:}, \mathbf{B}'_{t,<\ell} = \mathbf{b}'_{t,<\ell} \mid R' = r']}\right).\end{aligned}$$

Here $\mathbf{b}_{<t,:}$ denote the outputs of the SVT queries at times $1, \ldots, t-1$ and all levels, while $\mathbf{b}_{t,<\ell}$ denote the outputs of the SVT queries at time t but only for levels $0, \ldots, \ell-1$. The last equality holds because the internal threshold and query noise for each $\mathrm{SVT}^{j,w}$ instance are independent conditioned on $\mathbf{b}_{<t,:}$ and $\mathbf{b}_{t,<\ell}$.

Conditioned on identical outputs at past times and previous levels at the current time, the levels of all vertices are identical in both streams at the start of round t. For any vertex $v \notin \{x, y\}$, the induced degree $\deg^+_{X_j}(v)$ evaluates exactly the same on both streams because the differing edge e_{t^*} is not incident to v. This isolates the privacy loss strictly to the endpoints of the differing edge, x and y:

$$\mathcal{L} = \sum_{j \in Q} \sum_{w \in \{x,y\}} \sum_{t=1}^{T} \sum_{\ell=0}^{F-2} \ln \left(\frac{\Pr[B_{t,\ell}(j,w) = b_{t,\ell}(j,w) \mid \mathbf{B}_{<t,:} = \mathbf{b}_{<t,:}, \mathbf{B}_{t,<\ell} = \mathbf{b}_{t,<\ell} \mid R = r]}{\Pr[B'_{t,\ell}(j,w) = b_{t,\ell}(j,w) \mid \mathbf{B}'_{<t,:} = \mathbf{b}'_{<t,:}, \mathbf{B}'_{t,<\ell} = \mathbf{b}'_{t,<\ell} \mid R' = r']} \right).$$

For a fixed endpoint $w \in \{x, y\}$ and a fixed subgraph j, we can partition the temporal-level indices $(t, \ell) \in [T] \times \{0, \ldots, F-2\}$ into epochs $E_1, E_2, \ldots, E_{k(j,w)}$ for some $k(j, w)$. Each epoch consists of a sequence of failures ($b_{t,\ell}(j, w)$ is "below") terminated by a single success ($b_{t,\ell}(j, w)$ is "above"). Because a vertex can advance at most $F - 1$ times, there are at most F successes and thus $k(j, w) \leq F$. In other words, we can write

$$\mathcal{L} = \sum_{j \in Q} \sum_{w \in \{x,y\}} \sum_{i=1}^{k(j,w)} \sum_{t,\ell \in E_i} \ln \left(\frac{\Pr[B_{t,\ell}(j,w) = b_{t,\ell}(j,w) \mid \mathbf{B}_{<t,:} = \mathbf{b}_{<t,:}, \mathbf{B}_{t,<\ell} = \mathbf{b}_{t,<\ell}, R = r]}{\Pr[B'_{t,\ell}(j,w) = b_{t,\ell}(j,w) \mid \mathbf{B}'_{<t,:} = \mathbf{b}'_{<t,:}, \mathbf{B}'_{t,<\ell} = \mathbf{b}'_{t,<\ell}, R' = r']} \right).$$

By viewing each epoch as the output of a single-response SVT, the privacy loss for each epoch is bounded by $3\varepsilon_1$, following the guarantees of the single-response SVT (Theorem A.7). That is, for every $i \leq k(j, w)$,

$$\sum_{t,\ell \in E_i} \ln \left(\frac{\Pr[B_{t,\ell}(j,w) = b_{t,\ell}(j,w) \mid \mathbf{B}_{<t,:} = \mathbf{b}_{<t,:}, \mathbf{B}_{t,<\ell} = \mathbf{b}_{t,<\ell}, R = r]}{\Pr[B'_{t,\ell}(j,w) = b_{t,\ell}(j,w) \mid \mathbf{B}'_{<t,:} = \mathbf{b}'_{<t,:}, \mathbf{B}'_{t,<\ell} = \mathbf{b}'_{t,<\ell}, R' = r']} \right) \leq 3\varepsilon_1.$$

Hence the total privacy loss is at most $|Q| \cdot 2 \cdot F \cdot (3\varepsilon_1)$. We set the privacy budget for each SVT instance to $\varepsilon_1 = \frac{\varepsilon}{6|Q|F}$. This substitution concludes the proof. □

C.2. **Approximation and Space Guarantees.** Given our privacy guarantees, we now prove our approximation guarantees in this section. We first prove our concentration bound on the samples we obtain in our procedure. Specifically, we show that with high probability, the sampled edges give an approximately accurate estimate of $\deg^+_{X_j}(v)$. Our proof strategy is as follows. We use the proof of the approximation given by Dhulipala et al. (2022, Theorem 4.7). In order to use their theorem, we consider a hypothetical set of level data structures where we keep *all* of the edges in the graph. Note that we do not maintain these level data structures in our algorithm but only use them for the sake of analysis. Let this set of level data structures be denoted as $\mathcal{K}$. We place each vertex v in $\mathcal{K}$ on the exact same level as v in the level data structures within Algorithm 5.1. Then, we show that the vertices in $\mathcal{K}$ satisfy modified versions of Invariant 3 (Lemma C.3) and Invariant 4 (Lemma C.4) in the work of Dhulipala et al. (2022), which directly gives our approximation factor by a modified version of (Dhulipala et al., 2022, Theorem 4.7).

In the below proofs, let $\deg_{\mathcal{K}}(\ell, j, v)$ be the induced degree of v in the j-th level data structure within $\mathcal{K}$ consisting of all neighbors of v that are on level ℓ and higher. We prove the following lemmas for graphs G_t and G'_t which are formed after the first $t \in [T]$ updates. Intuitively, it says that if a vertex v has not been promoted past level ℓ, then its induced degree in the hypothetical full-graph level data structure $\mathcal{K}$ is not much larger than the target cutoff $(1 + \eta)^j$. This is because our sampling rate is high enough that the SVT would

have detected a degree exceeding the threshold and promoted v to a higher level. The additive error arises from the SVT noise and sampling variance.

Lemma C.3. *If vertex v is in level $\ell < F-1$, with high probability, in subgraph X_j after the levels are updated by Algorithm 5.2, Line 31, then, $\deg_{\mathcal{K}}(\ell, j, v) \leq (1+\eta)^j + O\left(\frac{\log^3 n}{\eta^2 \varepsilon}\right)$, with high probability.*

Proof. We prove this statement via contradiction. Suppose v is in level $\ell < F-1$ within subgraph X_j with high probability and $\deg_{\mathcal{K}}(\ell, j, v) > (1+\eta)^j + \frac{a_1 \log^3 n}{\varepsilon}$ with probability at least n^{-a_2} for some fixed constants $a_1, a_2 \geq 1$. We are in graph X_j, hence the probability we used to sample is $p_j = \frac{c_1 \log^3(n)}{\varepsilon(1+\eta)^j}$. The expected number of edges we sample out of the $\deg_{\mathcal{K}}(\ell, j, v)$ edges using p_j is at least

$$\mu \geq \frac{c_1 \log^3 n}{\varepsilon(1+\eta)^j} \cdot \left((1+\eta)^j + \frac{a_1 \log^3 n}{\varepsilon}\right) = \frac{c_1 \log^3 n}{\varepsilon} + \frac{a_1 c_1 \log^6 n}{\varepsilon^2 (1+\eta)^j} > \frac{c_1 \log^3 n}{\varepsilon}. \tag{C.1}$$

Let $S_{\ell,j,v}$ be the set of edges we sampled. Using a multiplicative Chernoff bound (Theorem A.1), we have that the probability that our sample has size smaller than $\frac{(1-\psi)c_1 \log^3 n}{\varepsilon}$ is as follows, for $\psi \in (0,1)$:

$$\mathbf{P}\left[|S_{\ell,j,v}| \leq (1-\psi)\mu\right] \leq \exp\left(-\frac{\mu\psi^2}{3}\right) \leq \exp\left(-\frac{\psi^2 c_1 \log^3 n}{3\varepsilon}\right). \tag{C.2}$$

The SVT introduces at most $\frac{a_3 \log^3(n)}{\varepsilon}$ additive error for some constant a_3, with high probability, and so the value we are comparing against the threshold of $p_j \cdot (1+\eta)^{j-1}$ is at least $|S_{\ell,j,v}| - \frac{2a_3 \log^3(n)}{\varepsilon}$, with high probability. We proved above that the probability that $|S_{\ell,j,v}| > \frac{(1-\psi)c_1 \log^3(n)}{\varepsilon}$ is at least $1 - \exp\left(-\frac{\psi^2 c_1 \log^3(n)}{3\varepsilon}\right)$. Thus, conditioned on the SVT error bound and sampling bound above, the SVT comparison succeeds if

$$|S_{\ell,j,v}| - \frac{2a_3 \log^3(n)}{\varepsilon} \geq \frac{(1-\psi)c_1 \log^3(n)}{\varepsilon} - \frac{2a_3 \log^3(n)}{\varepsilon} \tag{C.3}$$

$$\geq p_j \cdot (1+\eta)^{j-1} \tag{C.4}$$

$$= \frac{c_1 \log^3 n}{\varepsilon(1+\eta)}. \tag{C.5}$$

Note that this inequality always holds when

$$(1-\psi)c_1 - 2a_3 \geq \frac{c_1}{1+\eta}. \tag{C.6}$$

To satisfy the above expression, we require $\psi < \frac{\eta}{\eta+1}$ and $\eta > 0$, which is easily satisfied by the constraints of our problem, and also $c_1 \geq -\frac{(\eta+1)2a_3}{\eta(\psi-1)+\psi}$. We set c_1 to be an appropriately large enough constant in terms of η, a_3, ψ to amplify the probability of success to satisfy with high probability.

For an appropriate setting of c_1, ψ, a_3, we have that the SVT succeeds with probability at least $1 - n^{-c}$ for any constant $c \geq 1$. Taking the union bound over all levels $\ell' \leq \ell$ (such that SVT outputs succeed for all such ℓ'), this contradicts the fact that v is at level ℓ with high probability. □

Lemma C.3 directly shows that Invariant 3 is satisfied as in the work of Dhulipala et al. (2022). Now, we prove that Invariant 4 is also satisfied. The next lemma provides the complementary lower bound: if a vertex v has been promoted to level $\ell > 0$, then its degree in the level below must be large enough to justify the promotion. In other words, the SVT would not have accepted the promotion query unless the induced degree was close to the cutoff. Together with Lemma C.3, this pair of invariants ensures that the level assignments faithfully approximate the true core structure. The proof follows a similar structure to the proof of Lemma C.3.

Lemma C.4. *If vertex v is in level $\ell > 0$, with high probability, in subgraph X_j after the levels are updated by Algorithm 5.2, Line 31, then,* $\deg_{\mathcal{K}}(\ell-1, j, v) \geq (1+\eta)^{j-2} - O\left(\frac{\log^3 n}{\eta^2 \varepsilon}\right)$, *with high probability.*

Proof. As in the proof of Lemma C.3, we prove this lemma by contradiction. Suppose v is in level $\ell > 0$, with high probability, in subgraph X_j and $\deg_{\mathcal{K}}(\ell-1, j, v) < (1+\eta)^{j-2} - \frac{a_1 \log^3 n}{\varepsilon}$ with probability at least n^{-a_2} for some fixed constants $a_1, a_2 \geq 1$. Suppose that $\deg_{\mathcal{K}}(\ell, j, v) = (1+\eta)^{j-2} - \frac{a_1 \log^3 n}{\varepsilon} - 1$ since this is the worst case. Here, worst case means the probability that the SVT is satisfied and hence the vertex moves up a level is maximized. We are in graph X_j, hence the probability we used to sample is $\frac{c_1 \log^3(n)}{\varepsilon(1+\eta)^j}$. The expected number of edges we sample out of the $\deg_{\mathcal{K}}(\ell, j, v)$ edges using p_j is at least

$$\mu \geq \frac{c_1 \log^3 n}{\varepsilon(1+\eta)^j} \cdot \left((1+\eta)^{j-2} - \frac{a_1 \log^3 n}{\varepsilon} - 1\right) = \frac{c_1 \log^3 n}{\varepsilon(1+\eta)^2} - \frac{a_1 c_1 \log^6 n}{\varepsilon^2(1+\eta)^j} - p_j \tag{C.7}$$

$$\geq \frac{c_1 \log^3 n}{\varepsilon(1+\eta)^3} - \frac{2a_1 c_1 \log^6 n}{\varepsilon^2(1+\eta)^j}. \tag{C.8}$$

By Algorithm 5.2, Line 25, we set j such that $(1+\eta)^j \geq \frac{c_3 \log^3(n)}{\varepsilon}$. Hence, using this setting, we can further simplify Equation (C.8) as follows:

$$\frac{c_1 \log^3 n}{\varepsilon(1+\eta)^2} - \frac{2a_1 c_1 \log^6 n}{\varepsilon^2(1+\eta)^j} \geq \frac{c_1 \log^3 n}{\varepsilon(1+\eta)^2} - \frac{2a_1 c_1 \log^3(n)}{c_3 \varepsilon} = \left(\frac{1}{(1+\eta)^2} - \frac{2a_1}{c_3}\right) \frac{c_1 \log^3(n)}{\varepsilon}. \tag{C.9}$$

Let $S_{\ell,j,v}$ be the set of edges we sampled. Using a multiplicative Chernoff bound (Theorem A.1), we have that the probability that our sample has size larger than $(1+\psi)\mu$ is as follows, for $\psi \in (0,1)$:

$$\mathbf{P}\left[|S_{\ell,j,v}| \geq (1+\psi)\mu\right] \leq \exp\left(-\frac{\mu\psi^2}{3}\right) \leq \exp\left(-\frac{\psi^2\left(\frac{1}{(1+\eta)} - \frac{2a_1}{c_3}\right) c_1 \log^3 n}{3\varepsilon}\right). \tag{C.10}$$

The SVT introduces at most $\frac{a_3 \log^3(n)}{\varepsilon}$ additive error for some constant a_3, with high probability, and so the value we are comparing against the threshold of $p_j \cdot (1+\eta)^{j-1}$ is at most $|S_{\ell,j,v}| + \frac{2a_3 \log^3(n)}{\varepsilon}$, with high probability. We proved above that the probability that $|S_{\ell,j,v}| \leq \frac{(1+\psi)c_1 \log^3(n)}{\varepsilon(1+\eta)}$ is at least $1 - \exp\left(-\frac{\psi^2\left(\frac{1}{(1+\eta)} - \frac{2a_1}{c_3}\right) c_1 \log^3(n)}{3\varepsilon}\right)$. Conditioning on the

SVT error bound and the sampling bound above, the SVT fails the check at level $\ell - 1$ if

$$|S_{\ell,j,v}| + \frac{2a_3 \log^3(n)}{\varepsilon} \leq \frac{(1+\psi)c_1 \log^3(n)}{\varepsilon(1+\eta)^3} + \frac{2a_3 \log^3(n)}{\varepsilon} \tag{C.11}$$

$$< p_j \cdot (1+\eta)^{j-1} \tag{C.12}$$

$$= \frac{c_1 \log^3 n}{\varepsilon(1+\eta)}. \tag{C.13}$$

Note that the inequality is satisfied as long as we have

$$\frac{(1+\psi)c_1}{(1+\eta)^2} + 2a_3 < \frac{c_1}{1+\eta}. \tag{C.14}$$

To satisfy the above expression, we require $\psi < \eta$ and $\eta > 0$, which is easily satisfied by the constraints of our problem, and also $c_1 > -\frac{(\eta+1)^2 2a_3}{\psi-\eta}$. We set c_1 and c_3 to be appropriately large enough constants in terms of η, a_1, a_3, ψ to amplify the probability of success to satisfy with high probability.

For an appropriate setting of c_1, c_3, ψ, a_1, a_3, we have that the SVT failed the check at level $\ell - 1$ with probability at least $1 - n^{-c}$ for any constant $c \geq 1$. This contradicts the fact that v is at level ℓ, with high probability. □

For the formal proof of approximation guarantee, we also require the following folklore result.

Lemma C.5 (Folklore; (Matula and Beck, 1983))**.** *Suppose we perform the following* peeling *procedure. Provided an input graph $G = (V, E)$, we iteratively remove (peel) nodes with degree $\leq d^*$ for some $d^* > 0$ until no nodes with degree $\leq d^*$ remain. Then, the core number $k(v) \leq d^*$ for each removed node v and all remaining nodes w (not peeled) have core number $k(w) > d^*$.*

For Theorem 5.1, Lemma C.3 and Lemma C.4 together with arguments from Dhulipala et al. (2022, Theorem 4.7) complete our utility guarantees. For the sake of completeness, the next lemma explicitly mimics said arguments with the two invariants above to derive the overall approximation guarantee. Specifically, we simulate a peeling argument on the hypothetical full-graph data structure $\mathcal{K}$ and use the upper and lower bounds from Lemmas C.3 and C.4 to show that the estimated core number $\hat{k}(v)$ is within a $(2+\eta)$ multiplicative factor and an $O(\eta^{-2}\varepsilon^{-1}\log^3 n)$ additive error of the true core number $k(v)$ for every vertex.

Lemma C.6. *Algorithm 5.2 returns a $\left(2+\eta, O\left(\eta^{-2}\varepsilon^{-1}\log^3 n\right)\right)$-approximate k-core decomposition.*

Proof. Our proof follows almost verbatim from (Dhulipala et al., 2022, Theorem 4.7) except we use Lemma C.3 and Lemma C.4 instead. Since we return the same core estimate as the equivalent vertex in $\mathcal{K}$, we only need to show the approximation returned from $\mathcal{K}$ gives the correct approximation. In the below proof, a *group* g is defined as the g-th data structure in $\mathcal{K}$; hence the topmost level in the g-th group is the topmost level in the g-th level data structure in $\mathcal{K}$. We let Z_ℓ denote the set of nodes in levels $\geq \ell$ in any group g.

In this proof, when we refer to the level of a node i in subgraph j, we mean the level of any vertex $i \in V$ in $\mathcal{K}$. Using notation and definitions given by Dhulipala et al. (2022), let

$\hat{k}(i)$ be the core number estimate of i in $\mathcal{K}$ and $k(i)$ be the core number of i. First, we show that

$$\text{if } \hat{k}(i) \leq (2+\eta)(1+\eta)^{g'}, \text{ then } k(i) \leq (1+\eta)^{g'+1} + \frac{d\log^3 n}{\varepsilon} \tag{C.15}$$

with probability at least $1 - \frac{1}{n^{c-1}}$ for any group $g' \geq 0$, for any constant $c \geq 2$, and for appropriately large constant $d > 0$. Recall $F-1$ is the topmost level of group g'. In order for $(2+\eta)(1+\eta)^{g'}$ to be the estimate of i's core number, i must be in the topmost level of g' but below the topmost levels of all $g'' > g'$. Suppose we consider i's level in group $g'+1$ and let ℓ be i's level in $g'+1$. By Lemma C.3, if $\ell < F-1$, then $\deg_{\mathcal{K}}(\ell, g', i) \leq (1+\eta)^{g'} + O\left(\frac{\log^3 n}{\varepsilon}\right)$ with probability at least $1 - \frac{1}{n^c}$ for any constant $c \geq 1$. Furthermore, each node w at the same or lower level $\ell' \leq \ell$ has $\deg_{\mathcal{K}}(\ell', g', w) \leq (1+\eta)^{g'} + O\left(\frac{\log^3 n}{\varepsilon}\right)$, also by Lemma C.3.

Suppose we perform the following iterative procedure: starting from level $level = 0$ in g', remove all nodes in level $level$ during this turn and set $level \leftarrow level + 1$ for the next turn. Using this procedure, the nodes in level 0 are removed in the first turn, the nodes in level 1 are removed in the second turn, and so on until the graph is empty. Let $d_{level}(i)$ be the induced degree of any node i after the removal in the $(level-1)$-st turn and prior to the removal in the $level$-th turn. Since we showed that $\deg_{\mathcal{K}}(level, g', i) \leq (1+\eta)^{g'} + O\left(\frac{\log^3 n}{\varepsilon}\right)$ for any node i at level $level < F-1$, node i on level $level < F-1$ during the $level$-th turn has $d_{level}(i) \leq (1+\eta)^{g'} + O\left(\frac{\log^3 n}{\varepsilon}\right)$. Thus, when i is removed in the $level$-th turn, it has degree $\leq (1+\eta)^{g'} + O\left(\frac{\log^3 n}{\varepsilon}\right)$. Since all nodes removed before i also had degree $\leq (1+\eta)^{g'} + O\left(\frac{\log^3 n}{\varepsilon}\right)$ when they were removed, by Lemma C.5, node i has core number $k(i) \leq (1+\eta)^{g'} + O\left(\frac{\log^3 n}{\varepsilon}\right)$ with probability $\geq 1 - \frac{1}{n^c}$ for any $c \geq 1$. By the union bound over all nodes, this proves that $k(i) \leq (1+\eta)^{g'} + O\left(\frac{\log^3 n}{\varepsilon}\right)$ for all $i \in [n]$ with $\hat{k}(i) \leq (2+\eta)(1+\eta)^{g'}$ for all constants $c \geq 2$ with probability at least $1 - \frac{1}{n^{c-1}}$.

Now we prove our lower bound on $\hat{k}(i)$. We prove that for any $g' \geq 0$,

$$\text{if } \hat{k}(i) \geq (1+\eta)^{g'}, \text{ then } k(i) \geq \frac{(1+\eta)^{g'-2} - O\left(\frac{\log^3 n}{\varepsilon}\right)}{(2+\eta)(1+\eta)^2} \tag{C.16}$$

for all nodes i in the graph with probability at least $1 - \frac{1}{n^{c-1}}$ for any constant $c \geq 2$. We assume for contradiction that with probability $> \frac{1}{n^{c-1}}$ there exists a node i where $\hat{k}(i) \geq (1+\eta)^{g'}$ and $k(i) < \frac{(1+\eta)^{g'-2} - O\left(\frac{\log^3 n}{\varepsilon}\right)}{(2+\eta)(1+\eta)^2}$. This, in turn, implies that $k(i) \geq \frac{(1+\eta)^{g'-2} - O\left(\frac{\log^3 n}{\varepsilon}\right)}{(2+\eta)(1+\eta)^2}$ for all $i \in [n]$ where $\hat{k}(i) \geq (1+\eta)^{g'}$ holds with probability $< 1 - \frac{1}{n^{c-1}}$. To consider this case, we use the *pruning* process defined in Lemma C.5. In the below proof, let $d_S(i)$ denote the induced degree of node i in the subgraph induced by nodes in S. For a given subgraph S, we *prune* S by repeatedly removing all nodes $i \in S$ whose $d_S(i) < \frac{(1+\eta)^{g'-2} - \frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}$ for an appropriately selected constant $a > 0$. We consider levels from the same group g' since levels in groups lower than g' will also have smaller upper bound cutoffs, leading to an easier proof. Let j be the number of levels below level $F-1$. We prove via induction that the number of nodes pruned from the subgraph induced by Z_{F-1-j} (where Z_ℓ denotes the set of nodes in

levels $\geq \ell$ as defined earlier) must be at least

$$\left(\frac{(2+\eta)(1+\eta)^2}{2}\right)^{j-1}\left(\left((1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}\right)\left(1-\frac{1}{(2+\eta)(1+\eta)^2}\right)\right). \tag{C.17}$$

We first prove the base case when $j=1$. In this case, we know that for any node i in level $F-1$ in group g', it holds that $\deg_{\mathcal{K}}(F-1,g',i)\geq(1+\eta)^{g'-2}-O\left(\frac{\log^3 n}{\varepsilon}\right)$ with probability $\geq 1-\frac{1}{n^c}$ for any $c\geq 1$ by Lemma C.4. Taking the union bound over all nodes in level $F-1$ shows that this bound holds for all nodes $i\in[n]$ with probability at least $1-\frac{1}{n^{c-1}}$. All below expressions hold with probability at least $1-\frac{1}{n^{c-1}}$; for simplicity, we omit this phrase when giving these expressions. In order to prune i from the graph, we must prune at least

$$\left((1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}\right)-\frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}$$
$$=\left((1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}\right)\cdot\left(1-\frac{1}{(2+\eta)(1+\eta)^2}\right)$$

neighbors of i from Z_{F-2}. We must prune at least this many neighbors in order to reduce the degree of i to below the cutoff for pruning a vertex (as we show more formally below). In the case when $(1+\eta)^{g'}\leq 4(1+\eta)^2(\frac{a\log^3 n}{\varepsilon})$,[10] then our original approximation statement in the lemma is trivially satisfied because the core number is always non-negative and so even if $k(i)=0$, this is still within our additive approximation bounds. Hence, we only need to prove the case when $(1+\eta)^{g'}>4(1+\eta)^2\left(\frac{a\log^3 n}{\varepsilon}\right)$.

Then, if fewer than $\left((1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}\right)\left(1-\frac{1}{(2+\eta)(1+\eta)^2}\right)$ neighbors of i are pruned from the graph, then i is not pruned from the graph. If i is not pruned from the graph, then i is part of a $\left(\frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}\right)$-core (by Lemma C.5) and $k(i)\geq\frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}$ for all $i\in[n]$ where $\hat{k}(i)\geq(1+\eta)^{g'}$ with probability $\geq 1-\frac{1}{n^{c-1}}$, a contradiction with our assumption. Thus, it must be the case that there exists at least one i where at least $\left((1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}\right)\left(1-\frac{1}{(2+\eta)(1+\eta)^2}\right)$ neighbors of i are pruned in Z_{F-2}. For our induction hypothesis, we assume that at least the number of nodes as indicated in Equation (C.17) is pruned for j and prove this for $j+1$.

Each node w in levels $F-1-j$ and above has $d_{Z_{F-2-j}}(w)\geq(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}$ by Lemma C.4 (recall that all j levels below $F-1$ are in group g'). For simplicity of expression, we denote $J\triangleq\left((1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}\right)\left(1-\frac{1}{(2+\eta)(1+\eta)^2}\right)$. Then, in order to prune the $\left(\frac{(2+\eta)(1+\eta)^2}{2}\right)^{j-1}J$ nodes by our induction hypothesis, we must prune at least

$$\left(\frac{(2+\eta)(1+\eta)^2}{2}\right)^{j-1}J\cdot\left(\frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{2}\right) \tag{C.18}$$

edges where we "charge" the edge to the endpoint that is pruned last. (Note that we actually need to prune at least $\left((1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}\right)\cdot\left(1-\frac{1}{(2+\eta)(1+\eta)^2}\right)$ edges per pruned node as in

[10] 4 is an arbitrarily chosen large enough constant.

the base case but $\frac{\left((1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}\right)}{2}$ lower bounds this amount.) Each pruned node prunes fewer than $\frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}$ edges. Thus, using Equation (C.18), the number of nodes that must be pruned from Z_{F-2-j} is

$$\left(\frac{(2+\eta)(1+\eta)^2}{2}\right)^{j-1} J \cdot \frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{2\left(\frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}\right)} = \left(\frac{(2+\eta)(1+\eta)^2}{2}\right)^{j} J. \tag{C.19}$$

Equation (C.19) proves our induction step. Using Equation (C.17), the number of nodes that must be pruned from $Z_{F-1-2\log_{(2+\eta)(1+\eta)^2/2}(n)}$ is greater than n since $J \geq 1$ by our assumption that $(1+\eta)^{g'-2} > 4\left(\frac{a\log^3 n}{\varepsilon}\right)$:

$$\left(\frac{(2+\eta)(1+\eta)^2}{2}\right)^{2\log_{(2+\eta)(1+\eta)^2/2}(n)} \cdot J \geq n^2. \tag{C.20}$$

Thus, at $j = 2\log_{(2+\eta)(1+\eta)^2/2}(n)$, we run out of nodes to prune. We have reached a contradiction as we require pruning greater than n nodes with probability at least $\frac{1}{n^{c-1}} \cdot \left(1-\frac{1}{n^{c-1}}\right) > 0$ via the union bound over all nodes where $\hat{k}(i) \geq (1+\eta)^{g'}$ and using our assumption that with probability $> \frac{1}{n^{c-1}}$ there exists an $i \in [n]$ where $k(i) < \frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}$. This contradicts the fact that more than n nodes can be pruned with 0 probability.

From Equation (C.15), we can first obtain the inequality $k(i) \leq \hat{k}(i) + \frac{d\log^3 n}{\varepsilon}$ since this bounds the case when $\hat{k}(i) = (2+\eta)(1+\eta)^{g'}$; if $\hat{k}(i) < (2+\eta)(1+\eta)^{g'}$ then the largest possible value for $\hat{k}(i)$ is $(2+\eta)(1+\eta)^{g'-1}$ by our algorithm and we can obtain the tighter bound of $k(i) \leq (1+\eta)^{g'} + \frac{d\log^3 n}{\varepsilon}$. We can substitute $\hat{k}(i) = (2+\eta)(1+\eta)^{g'}$ since $(1+\eta)^{g'+1} < (2+\eta)(1+\eta)^{g'}$ for all $\eta \in (0,1)$ and $\eta > 0$. Second, from Equation (C.16), for any estimate $(2+\eta)(1+\eta)^g$, the largest g' for which this estimate has $(1+\eta)^{g'}$ as a lower bound is $g' = g + \lfloor \log_{(1+\eta)}(2+\eta) \rfloor \geq g + \log_{(1+\eta)}(2+\eta) - 1$. Substituting this g' into $\frac{(1+\eta)^{g'-2}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}$ results in

$$\frac{(1+\eta)^{g+\log_{(1+\eta)}(2+\eta)-3}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2} = \frac{\frac{(2+\eta)(1+\eta)^g}{(1+\eta)^3}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2} = \frac{\frac{\hat{k}(i)}{(1+\eta)^3}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^2}.$$

Thus, we can solve $k(i) \geq \frac{\frac{\hat{k}(i)}{(1+\eta)^3}-\frac{a\log^3 n}{\varepsilon}}{(2+\eta)(1+\eta)^4}$ and $k(i) \leq \hat{k}(i) + \frac{d\log^3 n}{\varepsilon}$ to obtain $k(i) - \frac{d\log^3 n}{\varepsilon} \leq \hat{k}(i) \leq ((2+\eta)(1+\eta)^5)k(i) + \frac{a(2+\eta)(1+\eta)^5\log^3 n}{\varepsilon}$. Simplifying, we obtain

$$k(i) - O(\varepsilon^{-1}\log^3 n) \leq \hat{k}(i) \leq (2+\eta)(1+\eta)^5 k(i) + O(\varepsilon^{-1}\log^3 n)$$

which is consistent with the definition of a $(2+\eta', O(\varepsilon^{-1}\log n))$-factor approximation algorithm of core numbers, for any constant $\eta' > 0$ and appropriately chosen constant $\eta' > \eta \in (0,1)$ that depends on η'. □

The following lemma bounds the total space usage of the k-core Algorithm 5.2. Since each of the $|Q|$ subgraphs has F levels and we sample $O(\varepsilon^{-1}\log^3 n)$ edges per vertex per level, a Chernoff bound yields the stated $\tilde{O}(n)$ bound.

Lemma C.7. *Algorithm 5.2 uses $O\left(\frac{n\log^5 n}{\eta^4\varepsilon}\right)$ space, with high probability.*

Proof. The proof follows from a Chernoff bound. We set the probability of sampling to $\frac{c_1\log^3 n}{\varepsilon(1+\eta)^j}$; we sample $O\left(\frac{\log^3 n}{\varepsilon}\right)$ edges per level, per node, for each of F levels in each of $|Q|$ graphs. Hence, in total over $F\cdot|Q|$ levels, we use $O\left(\frac{\log^5 n}{\varepsilon}\right)$ space per node, with high probability. □

Finally, combining Lemmas C.2 and C.6 and theorem C.7 yields the proof of Theorem 5.1.

Appendix D. Proofs of Theorems 6.1 and 6.2

We now restate and prove Theorems 6.1 and 6.2.

Theorem D.1 (Sublinear Space Private Densest Subgraph)**.** *Fix $\eta\in(0,1]$. Algorithm 6.2 is an ε-edge differentially private algorithm for the densest subgraph problem in the continual release model for insertion-only streams. The algorithm returns a set of vertices whose induced subgraph is a $\left(2+\eta, O(\eta^{-1}\varepsilon^{-1}\log^2(n))\right)$-approximation of the densest subgraph in G_t, with probability at least $1-1/\operatorname{poly}(n)$, for all $t\in[T]$. The maximum space used is $O\left(\eta^{-2}\varepsilon^{-1}n\log^2(n)\right)$, with probability at least $1-1/\operatorname{poly}(n)$, for all $t\in[T]$.*

Theorem D.2 (Sublinear Space Private Densest Subgraph)**.** *Fix $\eta\in(0,1]$. There exists an ε-edge differentially private algorithm for the densest subgraph problem in the continual release model for insertion-only streams. The algorithm returns a set of vertices whose induced subgraph is a $\left(1+\eta, O(\eta^{-4}\varepsilon^{-1}\log^5(n))\right)$-approximation of the densest subgraph in G_t, with probability at least $1-1/\operatorname{poly}(n)$, for all $t\in[T]$. The maximum space used is $O\left(\eta^{-5}\varepsilon^{-1}n\log^5(n)\right)$, with probability at least $1-1/\operatorname{poly}(n)$, for all $t\in[T]$.*

The pseudocode of our data structure and algorithm can be found in Algorithms 6.1 and 6.2. We also provide a detailed description in Section 6.1.

D.1. Privacy Guarantee.

In this section, we prove the privacy guarantees of our algorithm.

Lemma D.3. *Algorithm 6.2 is ε-DP in the continual release model on edge-neighboring insertion-only streams.*

Proof. Let $\mathcal{S}$ and $\mathcal{S}'$ be edge-neighboring insertion-only streams differing in exactly one update at time t^*, such that $e_{t^*}\neq e'_{t^*}$. At any timestep $t\in[T]$, the observable transcript $\mathcal{T}_t$ generated by the algorithm on input stream $\mathcal{S}$ consists of three random variables corresponding to the outputs of the internal differentially private mechanisms:

- $C_t\in\{$"above", "below"$\}$: The output of the edge-count Sparse Vector Technique (SVT).
- $D_t\in\{$"above", "below"$\}$: The output of the density SVT.
- S_t: The vertex set released by the static PrivateDSG subroutine (where $S_t=S_{t-1}$ if $D_t=$ "below").

Similarly, define C'_t, D'_t, S'_t on input $\mathcal{S}'$.

Let R, R' denote the random coins used to subsample edges in the two streams. We can couple R and R' to be identical on common edges $e_j = e'_j$ for $j \neq t^*$. Under this coupling (R, R'), the sampled graphs X_t and X'_t differ by at most the single edge e_{t^*}.

Define the privacy loss variable $\mathcal{L}$ by drawing outputs $(c_t, d_t, s_t) \sim (C_t, D_t, S_t)$, coupled random coins $(r, r') \sim (R, R')$, and computing the log-likelihood ratio of the transcript over the entire stream:

$$\begin{aligned}\mathcal{L} &:= \ln\left(\frac{\Pr[C_{1:T} = c_{1:T}, D_{1:T} = d_{1:T}, S_{1:T} = s_{1:T} \mid R = r]}{\Pr[C'_{1:T} = c_{1:T}, D'_{1:T} = d_{1:T}, S'_{1:T} = s_{1:T} \mid R' = r']}\right) \\ &= \sum_{t=1}^{T} \ln\left(\frac{\Pr[C_t = c_t, D_t = d_t, S_t = s_t \mid C_{<t} = c_{<t}, D_{<t} = d_{<t}, S_{<t} = s_{<t}, R = r]}{\Pr[C'_t = c_t, D'_t = d_t, S'_t = s_t \mid C'_{<t} = c_{<t}, D'_{<t} = d_{<t}, S'_{<t} = s_{<t}, R' = r']}\right).\end{aligned}$$

Recall that an algorithm is ε-DP if and only if the absolute value of the privacy loss variable is bounded above by ε with probability 1 (Dwork and Roth, 2014, Lemma 3.17).

Using the chain rule of conditional probability, we expand $\mathcal{L}$ across all timesteps and decompose it into the three sequential mechanism evaluations:

$$\begin{aligned}\mathcal{L} &= \sum_{t=1}^{T} \ln\left(\frac{\Pr[C_t = c_t \mid C_{<t} = c_{<t}, D_{<t} = d_{<t}, S_{<t} = s_{<t}, R = r]}{\Pr[C'_t = c_t \mid C'_{<t} = c_{<t}, D'_{<t} = d_{<t}, S'_{<t} = s_{<t}, R' = r']}\right) \\ &\quad + \sum_{t=1}^{T} \ln\left(\frac{\Pr[D_t = d_t \mid C_{\leq t} = c_{\leq t}, D_{<t} = d_{<t}, S_{<t} = s_{<t}, R = r]}{\Pr[D'_t = d_t \mid C'_{\leq t} = c'_{\leq t}, D'_{<t} = d'_{<t}, S'_{<t} = s'_{<t}, R' = r']}\right) \\ &\quad + \sum_{t=1}^{T} \ln\left(\frac{\Pr[S_t = s_t \mid C_{\leq t} = c_{\leq t}, D_{\leq t} = d_{\leq t}, S_{<t} = s_{<t}, R = r]}{\Pr[S'_t = s_t \mid C'_{\leq t} = c'_{\leq t}, D'_{\leq t} = d'_{\leq t}, S'_{<t} = s'_{<t}, R' = r']}\right) \\ &= \sum_{t=1}^{T} \mathcal{L}_{t,C} + \sum_{t=1}^{T} \mathcal{L}_{t,D} + \sum_{t=1}^{T} \mathcal{L}_{t,S}.\end{aligned}$$

We will upper bound each summation separately.

Edge-Count SVT Loss. Let $t_1^C < t_2^C < \cdots < t_{k_C}^C$ be the timesteps where the edge-count SVT outputs "above". We group the summation into epochs:

$$\sum_{t=1}^{T} \mathcal{L}_{t,C} = \sum_{i=1}^{k_C} \sum_{t=t_{i-1}^C+1}^{t_i^C} \ln\left(\frac{\Pr[C_t = c_t \mid C_{<t} = c_{<t}, D_{<t} = d_{<t}, S_{<t} = s_{<t}, R = r]}{\Pr[C'_t = c_t \mid C'_{<t} = c_{<t}, D'_{<t} = d_{<t}, S'_{<t} = s_{<t}, R' = r']}\right).$$

Conditioned on the identical past, the internal threshold m' is identical in both streams. Since $\mathcal{S}$ and $\mathcal{S}'$ differ by exactly one update, the global sensitivity is $\Delta = 1$. By viewing this as a simple composition over single-response SVT mechanisms with a total privacy budget of $\varepsilon_2 = \varepsilon/3$, SVT guarantees (Theorem A.7) that with probability 1:

$$\sum_{t=1}^{T} \mathcal{L}_{t,C} \leq \frac{\varepsilon}{3}.$$

Density SVT Loss. Let $t_1^D < t_2^D < \cdots < t_{k_D}^D$ be the timesteps where the density SVT outputs "above". We again group the summation into epochs:

$$\sum_{t=1}^{T} \mathcal{L}_{t,D} = \sum_{i=1}^{k_D} \sum_{t=t_{i-1}^D+1}^{t_i^D} \ln\left(\frac{\Pr[D_t = d_t \mid C_{\le t} = c_{\le t}, D_{<t} = d_{<t}, S_{<t} = s_{<t}, R = r]}{\Pr[D'_t = d_t \mid C'_{\le t} = c'_{\le t}, D'_{<t} = d'_{<t}, S'_{<t} = s'_{<t}, R' = r']}\right).$$

Conditioned on an identical past, the sampling probability p_t is a deterministic function of the public transcript and is therefore identical on neighboring streams. Under the coupling (R, R'), the exact density query D_t on X_t has a sensitivity of 1. Similar to the edge count, this query sequence is processed by the second SVT with a budget of $\varepsilon_2 = \varepsilon/3$. It follows that:

$$\sum_{t=1}^{T} \mathcal{L}_{t,D} \le \frac{\varepsilon}{3}.$$

Static DP Subroutine Loss. Finally, the algorithm only invokes the static ε_1-DP PrivateDSG subroutine at time t if $d_t =$ "above". Therefore, $\mathcal{L}_{t,S} = 0$ for all $t \notin \{t_1^D, t_2^D, \ldots, t_{k_D}^D\}$ and

$$\sum_{t=1}^{T} \mathcal{L}_{t,S} = \sum_{i=1}^{k_D} \ln\left(\frac{\Pr[S_{t_i^D} = s_{t_i^D} \mid C_{\le t_i^D} = c_{\le t_i^D}, D_{\le t_i^D} = d_{\le t_i^D}, S_{<t_i^D} = s_{<t_i^D}, R = r]}{\Pr[S_{t_i^D} = s_{t_i^D} \mid C'_{\le t_i^D} = c'_{\le t_i^D}, D'_{\le t_i^D} = d'_{\le t_i^D}, S_{<t_i^D} = s'_{<t_i^D}, R' = r']}\right).$$

Conditioned on an identical past, the sampled graph X_t differs by at most 1 edge between the two streams, so each call incurs a privacy loss of ε_1. The maximum number of times the density SVT can output "above" is capped at $k_D \le c_2 \log_{1+\eta}(n)$. By sequential composition:

$$\sum_{t=1}^{T} \mathcal{L}_{t,S} \le k_D \cdot \varepsilon_1 \le c_2 \log_{1+\eta}(n) \cdot \left(\frac{\varepsilon}{3c_2 \log_{1+\eta}(n)}\right) = \frac{\varepsilon}{3}.$$

Conclusion. Summing the upper bounds of the decomposed components over the entire stream, the absolute value of the total privacy loss variable $\mathcal{L}$ is almost surely bounded by:

$$\mathcal{L} \le \frac{\varepsilon}{3} + \frac{\varepsilon}{3} + \frac{\varepsilon}{3} = \varepsilon.$$

This confirms that the algorithm is ε-edge differentially private in the continual release model. □

Remark D.4. We remark that although we are running a DP algorithm on a sample which may imply some privacy amplification bounds, these privacy savings only apply to our case if the sampling probability of the edges decreases per time step. Since we cannot guarantee that this will be the case for all input graphs, the privacy amplification bounds do not improve the worst-case privacy guarantees.

D.2. **Approximation and Space Guarantees.** In this section, we prove the approximation factor of the approximate densest subgraph we obtain via our sampling procedure. We first prove that our sampling procedure produces, with high probability, a set of edges $X = X_t \subseteq E_t$ at time t such that $|\mathrm{OPT}(G_t) - \mathrm{den}_{G_t}(\mathrm{OPT}(X_t))| \leq \eta \cdot \mathrm{OPT}(G_t)$ where $\mathrm{OPT}(G_t)$ is the density of the densest subgraph in the dynamic graph G_t at time t and $\mathrm{den}_{G_t}(\mathrm{OPT}(X_t))$ is the density of the densest subgraph in the sampled graph $G[X_t]$ but with respect to the full graph G_t. Then, we show that the approximate solution returned by the private static densest subgraph subroutine applied to the sub-sampled graph translates to an approximate solution in the original graph.

In the following, we assume full independence in each of the sampled values for each edge (which is guaranteed by our algorithm). The next lemma is the main approximation guarantee for our densest subgraph algorithm. It shows that the vertex set V^* returned by the algorithm induces a subgraph whose density in G_t is within a $(2+\eta)$ multiplicative factor and an $O(\varepsilon^{-1}\eta^{-1}\log^2 n)$ additive error of the true optimum. The proof proceeds in two parts: first, we show that sub-sampling preserves the density of every induced subgraph up to a $(1+\eta)$ factor via a Chernoff bound and union bound over all vertex subsets; second, we compose this with the static private densest subgraph subroutine to obtain the overall approximation.

Lemma D.5. *Fix $\eta \in (0,1]$. For every $t \in [T]$, Algorithm 6.2 returns a set of vertices $V^* \subseteq V$ where the induced subgraph $G_t[V^*]$ has density $\mathrm{den}_{G_t}(V^*) \geq \frac{\mathrm{den}_{G_t}(V_{\mathrm{OPT}})}{(2+\eta)} - O\left(\frac{\log^2 n}{\varepsilon\eta}\right)$ (where V_{OPT} is the set of vertices in a densest subgraph at time t consisting of all edges seen in the stream so far), with high probability.*

Proof. For the first part of this proof, we take inspiration from the techniques given by Esfandiari et al. (2016, Lemma 2.3), although our proof differs in several respects due to the noise resulting from our private mechanisms.

Sub-Sampling Concentration. Let x_e be the random variable indicating whether edge e exists in X and p be the probability of sampling the edges. Fix an arbitrary non-empty subset of vertices $\varnothing \neq V' \subseteq V$. The number of edges in $X[V']$ is given by $|X[V']| = \sum_{e \in X[V']} 1 = \sum_{e \in G_t[V']} x_e$. We use G_t to denote the graph on edges inserted in updates $u_1, \ldots, u_t$ at timestep t. Then, we know the expectation of $\mathrm{den}(X[V'])$ is

$$\mu = \mathbb{E}[\mathrm{den}(X[V'])] = \mathbb{E}\left[\frac{\sum_{e \in G_t[V']} x_e}{|V'|}\right] = \frac{\mathbb{E}\left[\sum_{e \in G_t[V']} x_e\right]}{|V'|} = \frac{\sum_{e \in G_t[V']} p}{|V'|} \tag{D.1}$$

$$= p \cdot \frac{\sum_{e \in G_t[V']} 1}{|V'|} = p \cdot \mathrm{den}_{G_t}(V'), \tag{D.2}$$

and the expectation of $|X[V']|$ is also

$$\mu_{|X[V']|} = \mathbb{E}[|X[V']|] = \mathbb{E}\left[\sum_{e \in G_t[V']} x_e\right] = p \cdot |G_t[V']|, \tag{D.3}$$

where $|G_t[V']|$ denotes the number of edges in the induced subgraph of $G_t[V']$.

Since all the variables x_e are independent, we have by the multiplicative Chernoff bound (Theorem A.1),

$$\mathbf{P}\left[|X[V']| \geq (1+\psi)\cdot\mu_{|X[V']|}\right] \leq \exp\left(-\frac{\psi^2\mu_{|X[V']|}}{2+\psi}\right) \tag{D.4}$$

$$\leq \exp\left(-\frac{\psi^2\mu_{|X[V']|}}{14}\right) \qquad \text{When } \psi\in(0,12] \tag{D.5}$$

$$\leq \exp\left(-\frac{\psi^2\cdot p\cdot|G_t(V')|}{14}\right). \tag{D.6}$$

Consider the private estimate m'_t of m_t stored in Algorithm 6.1, Line 3, where m_t is the number of true edges in the graph. We add $O(T)$ instances of $\mathsf{Lap}(O(\varepsilon/\log_{1+\eta}(n)))$ noise in the SVT subroutine (Algorithm 6.2, Line 29). By a union bound over $T=\text{poly}(n)$, all such noises are of order $O(\varepsilon^{-1}\log_{1+\eta}(n)\log(n)) = O(\varepsilon^{-1}\eta^{-1}\log^2(n))$ with probability $1-1/\text{poly}(n)$. Hence $m_t - O\left(\frac{\log^2 n}{\varepsilon\eta}\right) \leq m'_t \leq (1+\eta)\cdot m_t + O\left(\frac{\log^2 n}{\varepsilon\eta}\right)$, with high probability. Note that by the choice of initialization of m' (Algorithm 6.1, Line 3), we sub-sample edges only once $m_t \geq \Omega(\varepsilon^{-1}\eta^{-2}n\log^2(n)) - O(\varepsilon^{-1}\eta^{-1}\log^2(n)) = \omega(\varepsilon^{-1}\eta^{-1}\log^2(n))$. Thus, we also know that (by Algorithm 6.1, Line 4), for a constant $c>0$,

$$1 \geq p = \frac{cn\log^2(n)}{\varepsilon\cdot\eta^2\cdot m'_t} \tag{D.7}$$

$$\geq \frac{cn\log^2(n)}{\varepsilon\cdot\eta^2\cdot\left((1+\eta)\cdot m_t + O\left(\frac{\log^2 n}{\varepsilon\eta}\right)\right)} \tag{D.8}$$

$$\geq \frac{cn\log^2(n)}{c'\left(\varepsilon\cdot\eta^2\cdot m_t\right)} \qquad \text{since } m_t = \omega(\varepsilon^{-1}\eta^{-1}\log^2 n) \tag{D.9}$$

for an appropriately large constant $c'>0$.

Let $k=|V'|$. We set $\psi = \frac{pk\eta\text{OPT}(G_t)}{2\mu_{|X[V']|}}$ in Equation (D.6) where $\text{OPT}(G_t)$ is the density of the densest subgraph in G_t. For the analysis, we consider two cases. The first case is when $\psi\leq 12$ and the second is when $\psi>12$.[11] Suppose in the first case that $\psi\leq 12$; substituting this value for ψ into Equation (D.4) yields the following bound

$$\mathbf{P}\left[|X[V']| \geq (1+\psi)\cdot\mu_{|X[V']|}\right] \tag{D.10}$$

$$\leq \exp\left(-\frac{p^2k^2\eta^2\text{OPT}(G_t)^2\cdot\mu_{|X[V']|}}{14\cdot 4\mu^2_{|X[V']|}}\right) \tag{D.11}$$

$$\leq \exp\left(-\frac{pk\eta^2\text{OPT}(G_t)}{56}\right), \tag{D.12}$$

where the last expression follows since $p\cdot k\cdot\text{OPT}(G_t) \geq \mu_{|X[V']|}$ due to the fact that by Equation (D.3), $p\cdot k\cdot\text{OPT}(G_t) \geq p\cdot k\cdot\text{den}(G[V']) = p\cdot(|G[V']|) = \mu_{|X[V']|}$. Then, suppose that $\psi>12$; substituting this ψ into Theorem A.1 gives the following probability expression: $\mathbf{P}[X\geq(1+\psi)\mu] \leq \exp\left(-\frac{\psi\mu}{2}\right)$ since $\frac{\psi}{2+\psi}\geq\frac{1}{2}$ for the given bound on ψ. Thus,

[11]The two cases selected are bounded by 12 which is an arbitrarily chosen large enough constant.

using this value of ψ in Equation (D.4) yields the following bound

$$\mathbf{P}\left[|X[V']| \geq (1+\psi)\cdot\mu_{|X[V']|}\right] \tag{D.13}$$

$$\leq \exp\left(-\frac{pk\eta\mathrm{OPT}(G_t)\cdot\mu_{|X[V']|}}{2\cdot 2\mu_{|X[V']|}}\right) \tag{D.14}$$

$$\leq \exp\left(-\frac{pk\eta\mathrm{OPT}(G_t)}{4}\right). \tag{D.15}$$

Altogether, for both cases, our expression is bounded by

$$\mathbf{P}\left[|X[V']| \geq (1+\psi)\cdot\mu_{|X[V']|}\right] \leq \exp\left(-\frac{pk\eta^2\mathrm{OPT}(G_t)}{56}\right). \tag{D.16}$$

We can simplify the LHS of the above inequality by:

$$\mathbf{P}\left[|X[V']| \geq \mu_{|X[V']|} + \frac{pk\eta\mathrm{OPT}(G_t)}{2}\right] = \mathbf{P}\left[\frac{|X[V']|}{pk} \geq \frac{\mu_{|X[V']|}}{pk} + \frac{\eta\mathrm{OPT}(G_t)}{2}\right] \tag{D.17}$$

$$= \mathbf{P}\left[\frac{1}{p}\cdot\mathrm{den}(X[V']) \geq \frac{1}{p}\cdot p\cdot\mathrm{den}_{G_t}(V') + \frac{\eta\mathrm{OPT}(G_t)}{2}\right] \tag{D.18}$$

$$= \mathbf{P}\left[\frac{1}{p}\cdot\mathrm{den}(X[V']) \geq \mathrm{den}_{G_t}(V') + \frac{\eta\mathrm{OPT}(G_t)}{2}\right]. \tag{D.19}$$

We now consider all subsets of vertices of size k. There are at most $\binom{n}{k}$ such sets. There are also n possible values of k since there are at most n vertices. Hence, the total probability that the bound holds for all $V' \subseteq V$ (using Equation (D.16)) is

$$\sum_{k=1}^{n}\binom{n}{k}\cdot\exp\left(-\frac{pk\eta^2\mathrm{OPT}(G_t)}{56}\right) \leq \sum_{k=1}^{n} n^k\cdot\exp\left(-\frac{pk\eta^2\mathrm{OPT}(G_t)}{56}\right) \tag{D.20}$$

$$= \sum_{k=1}^{n}\exp\left(k\cdot\ln(n) - \frac{pk\eta^2\mathrm{OPT}(G_t)}{56}\right). \tag{D.21}$$

Note that the densest subgraph has density at least that of the entire graph, $\frac{m_t}{n}$. Combined with the substitution of Equation (D.9) into the above, we obtain

$$\sum_{k=1}^{n} \exp\left(k \cdot \ln(n) - \frac{pk\eta^2 \mathrm{OPT}(G_t)}{56}\right) \tag{D.22}$$

$$\leq \sum_{k=1}^{n} \exp\left(k \cdot \ln(n) - \frac{cn\log^2(n)}{c'(\varepsilon \cdot \eta^2 \cdot m_t)} \cdot \frac{k\eta^2 m_t}{56n}\right) \qquad \text{by Equation (D.9)} \tag{D.23}$$

$$\leq \sum_{k=1}^{n} \exp\left(k \cdot 2\log^2(n) - \frac{c\log^2(n)}{c'\varepsilon} \cdot \frac{k}{56}\right) \tag{D.24}$$

$$= \sum_{k=1}^{n} \exp\left(-\left(\frac{c}{56c'\varepsilon} - 2\right) \cdot k \cdot \log^2(n)\right) \tag{D.25}$$

$$\leq \sum_{k=1}^{n} \exp\left(-\left(\frac{c}{56c'\varepsilon} - 2\right) \cdot \log^2(n)\right) \qquad \text{since } k \geq 1 \tag{D.26}$$

$$= n \cdot \exp\left(-\left(\frac{c}{56c'\varepsilon} - 2\right) \cdot \log^2(n)\right) \tag{D.27}$$

$$\leq \exp\left(-\left(\frac{c}{56c'\varepsilon} - 4\right) \cdot \log^2(n)\right) \tag{D.28}$$

where Equation (D.26) holds when $\left(\frac{c}{56c'\varepsilon} - 2\right) \geq 1$. To obtain the high probability bound of $1/\operatorname{poly}(n)$ over all timesteps t, it suffices to set c to be a sufficiently large absolute constant and take a union bound over $T = \operatorname{poly}(n)$ events. Hence, we have proven that with high probability, the density of any subgraph $G_t[V']$ is at least $\frac{1}{p}\operatorname{den}_X(V') - \frac{\eta}{2}\mathrm{OPT}(G_t)$ for every t. In particular, no induced subgraph in X has estimated density greater than $(1 + \frac{\eta}{2})p \cdot \mathrm{OPT}(G_t)$, with high probability.

Now, we show that $\mathrm{OPT}(G_t)$ has large enough induced density in X such that the returned subgraph is a $(1+\eta)$-approximate densest subgraph in G_t. We do this by setting $V' = V(\mathrm{OPT}(G_t))$, i.e. to the set of vertices of the densest subgraph in G_t. By applying a Chernoff bound (Theorem A.1) with $\psi = \eta \in (0, 1]$ and using the fact that $\mathrm{OPT}(G_t) \geq {}^{m_t}/_{n}$, we have that

$$\mathbf{P}\left[\frac{1}{p}\operatorname{den}(X[V']) \leq (1-\psi) \cdot \mathrm{OPT}(G_t)\right] \tag{D.29}$$

$$= \mathbf{P}\left[\operatorname{den}(X[V']) \leq (1-\psi) \cdot \mu\right] \tag{D.30}$$

$$\leq \exp\left(-\frac{\psi^2\mu}{2}\right) = \exp\left(-\frac{\psi^2 \cdot p \cdot \mathrm{OPT}(G_t)}{2}\right) \tag{D.31}$$

$$\leq \exp\left(-\frac{\psi^2}{2} \cdot \frac{cn\log^2(n)}{c'(\varepsilon \cdot \eta^2 \cdot m_t)} \cdot \frac{m_t}{n}\right) \qquad \text{by Equation (D.9)} \tag{D.32}$$

$$= \exp\left(-\frac{c\log^2(n)}{2c'\varepsilon}\right) \qquad \text{by setting } \psi = \eta. \tag{D.33}$$

By setting a large enough constant $c > 0$ and together with what we showed above, we obtain that with high probability, the densest subgraph in X has induced density in G_t that is a $(1+\eta)$-approximation of $\mathrm{OPT}(G_t)$. We take the union bound over $T = \operatorname{poly}(n)$ to show that for each time step t, our bound holds.

Approximation. By Theorem G.1, the vertex set S we output is a $(2, O(\varepsilon_1^{-1}\log n))$-approximate densest subgraph of X after each call to the private static densest subgraph subroutine. If we have yet to begin sub-sampling, i.e. $X = G_t$, then we are done.

Suppose the algorithm started to sub-sample. Then with high probability, after each call to the private static densest subgraph algorithm,

$$\begin{aligned}
&\operatorname{den}_{G_t}(S) && && \text{(D.34)}\\
&\geq \frac{1}{p}\operatorname{den}_X(S) - \frac{\eta\operatorname{OPT}(G_t)}{2} && \text{by Equation (D.19)} && \text{(D.35)}\\
&\geq \frac{1}{p}\left(\frac{\operatorname{OPT}(X)}{2} - O(\varepsilon_1^{-1}\log n)\right) - \frac{\eta\operatorname{OPT}(G_t)}{2} && \text{by Theorem G.1} && \text{(D.36)}\\
&\geq \frac{(1-\eta)\operatorname{OPT}(G_t)}{2} - \frac{c'\left(\varepsilon\cdot\eta^2\cdot m_t\right)}{cn\log^2(n)}\cdot\frac{c''\log^2(n)}{\varepsilon\eta} - \frac{\eta\operatorname{OPT}(G_t)}{2} && \text{since } \varepsilon_1 = \Theta\left(\frac{\varepsilon}{\log_{1+\eta}(n)}\right) && \text{(D.37)}\\
&= \frac{(1-2\eta)\operatorname{OPT}(G_t)}{2} - \frac{c'c''\eta m_t}{cn} && && \text{(D.38)}\\
&\geq \frac{(1-3\eta)\operatorname{OPT}(G_t)}{2}, && \text{for } c \geq 2c'c'', \operatorname{OPT}(G_t) \geq \frac{m_t}{n} && \text{(D.39)}
\end{aligned}$$

where Equation (D.37) follows from Equation (D.29).

Finally, we account for the additional error due to not updating S when the SVT (Algorithm 6.2, Line 36) does not output "above". Let D_t denote the exact density of the subgraph and p_t the sampling probability at time t as returned on Algorithm 6.2, Line 35. If the SVT did not output "above", we know that with high probability,

$$\begin{aligned}
D_t &\leq p_t\cdot L + O(\varepsilon^{-1}\eta^{-1}\log^2(n)) && \text{(D.40)}\\
\frac{D_t}{p_t} &\leq L + \frac{O(\varepsilon^{-1}\eta^{-1}\log^2(n))}{p_t} && \text{(D.41)}
\end{aligned}$$

$$\begin{aligned}
D_t &> p_t\cdot\frac{L}{1+\eta} - O(\varepsilon^{-1}\eta^{-1}\log^2(n)) && \text{(D.42)}\\
\frac{D_t}{p_t} &> \frac{L}{1+\eta} - \frac{O(\varepsilon^{-1}\eta^{-1}\log^2(n))}{p_t}. && \text{(D.43)}
\end{aligned}$$

Similar to the calculation we made regarding $\operatorname{OPT}(X)$, either $p = 1$ and there is nothing to prove or $p \geq \frac{cn\log^2(n)}{c'\varepsilon\eta^2 m_t}$ and

$$\begin{aligned}
\frac{O(\varepsilon^{-1}\eta^{-1}\log^2(n))}{p_t} &= O(\varepsilon^{-1}\eta^{-1}\log^2(n))\cdot\frac{c'\varepsilon\eta^2 m_t}{cn\log^2(n)} && \text{(D.44)}\\
&\leq \frac{\eta m_t}{n} && \text{(D.45)}\\
&\leq \eta\operatorname{OPT}(G_t). && \text{(D.46)}
\end{aligned}$$

In other words, assuming a sufficiently large constant c, we incur an additional $(1+2\eta)$ multiplicative error and $O(\varepsilon^{-1}\eta^{-1}\log^2 n)$ additive error due to the SVT. Hence, we obtain a $(2+O(\eta), O(\varepsilon^{-1}\eta^{-1}\log^2 n))$-approximate densest subgraph at every timestamp t. □

Combining Lemmas D.3 and D.5 proves the privacy and utility guarantees of Theorem 6.1. Before addressing the space usage, we briefly describe the minor changes needed to obtain similar guarantees for Theorem 6.2.

Reducing the Multiplicative Factor. We remark that we require $1/p = \Omega(\varepsilon^{-1}\eta^{-2}n\log^2 n)$ to absorb the additive error of the private static densest subgraph subroutine (Theorem G.1) into the multiplicative error (see Equation (D.37)). If we instead use an alternative private static densest subgraph subroutine, say Theorem G.2, which yields a $O(1+\eta, \eta^{-3}\varepsilon_1^{-1}\log^4(n))$-approximation, we would need to set $1/p = \Omega(\varepsilon^{-1}\eta^{-5}n\log^5(n))$ (Algorithm 6.1, Line 4) since $\varepsilon_1 = \varepsilon/\log_{1+\eta}(n)$. Suppose we further adjust the initial edge threshold to $m' = \Theta(\varepsilon^{-1}\eta^{-5}n\log^5(n))$ (Algorithm 6.1, Line 3), the initial density threshold to $L = \Theta(\varepsilon^{-1}\eta^{-5}n\log^5(n))$, and repeat the proof steps of Lemma D.5 for this version of our algorithm. This yields a proof of Theorem 6.2.

Space Complexity. Finally, the following lemma bounds the space usage of our data structure. Because the SVT controls the edge threshold m' and the sampling probability p is set inversely proportional to m', the expected number of sampled edges is $O(n\log^2(n)/(\varepsilon\eta^2))$ at any point in time. A Chernoff bound then yields the high-probability guarantee. The precise constant depends on which static private densest subgraph subroutine is used.

Lemma D.6. *Algorithm 6.1 uses $O(\varepsilon^{-1}\eta^{-2}n\log^2(n))$ (resp. $O(\varepsilon^{-1}\eta^{-5}n\log^5(n))$)[12] space with probability at least $1-1/\operatorname{poly}(n)$.*

Proof. By Algorithm 6.1, Line 4, each edge is sampled with probability $p = \min\left(1, \frac{c_4 n\log^2(n)}{\varepsilon\cdot\eta^2 m'}\right)$. By the guarantees of SVT and our scaling procedure in Algorithm 6.1, Line 3, the final m' will be at least $\frac{m}{1+\eta} - O(\log^2(n)/\varepsilon)$. Since we initially set $m' = \frac{c_3 n\log^2(n)}{\varepsilon\eta^2} \gg c_5\log^2(n)/\varepsilon$ (or $m' = \frac{c_3 n\log^5(n)}{\varepsilon\eta^2} \gg c_5\log^2(n)/\varepsilon$ depending on which static algorithm we choose), it follows that $m' \geq m/3$, with high probability. Then, the expected number of edges we sample is $\frac{3c_3 n\log^2(n)}{\varepsilon\eta^2}$ (resp. $\frac{3c_3 n\log^5(n)}{\varepsilon\eta^2}$). Since all edges are sampled independently, we use the Chernoff bound to show that the total space we use is $O(\varepsilon^{-1}\eta^{-2}n\log^2(n))$ (resp. $O(\varepsilon^{-1}\eta^{-2}n\log^5(n))$) with probability at least $1-1/\operatorname{poly}(n)$. □

Appendix E. Proof of Theorem 7.1

We now prove Theorem 7.1, whose statement we copy below for convenience.

Theorem E.1. *Fix $\eta \in (0,1]$. Given a public estimate $\tilde{\alpha}$ of the maximum arboricity α over the stream,[13] Algorithm 7.1 is an ε-edge DP algorithm for estimating the size of the maximum*

[12] This space usage depends on which static algorithm we use.

[13] We can eliminate this assumption with an additional pass of the stream or notify the observer when the utility guarantees no longer hold in the one-pass setting. See Appendix I.

matching in the continual release model for insertion-only streams. If $\tilde{\alpha} \geq \alpha$, then with probability at least $1 - 1/\operatorname{poly}(n)$, our algorithm returns a $\big((1+\eta)(2+\tilde{\alpha}), O(\eta^{-1}\varepsilon^{-1}\log^2(n))\big)$-approximation of the size of the maximum matching at every timestamp. Moreover, our algorithm uses $O(\eta^{-2}\varepsilon^{-1}\log^2(n)\log(\tilde{\alpha}))$ space with probability $1 - 1/\operatorname{poly}(n)$.

The pseudocode can be found in Algorithm 7.1 and we present a detailed description in Section 7.1.

E.1. **Privacy Guarantee.** We prove the privacy guarantees of our algorithm in this section.

Lemma E.2. *Algorithm 7.1 is ε-edge differentially private.*

Proof. Let $\mathcal{S} = (e_1, \ldots, e_T)$ and $\mathcal{S}' = (e'_1, \ldots, e'_T)$ be edge-neighboring insertion-only streams. Without loss of generality, they differ only at a single time $t^* \in [T]$, where $e_{t^*} = \{u, v\}$ and $e'_{t^*} = \perp$. For input $\mathcal{S}$ and each time t, let A_t be the random variable representing the entire answer string returned by EstimateSVT during the while-loop in Algorithm 7.1, Line 18 at time t (that is, all answers produced while processing the repeated threshold queries at time t), and let B_t be the single answer returned by SubsampleSVT in Algorithm 7.1, Line 23 at time t. Similarly, define A'_t, B'_t for input $\mathcal{S}'$.

Let R, R' denote the set of random coins used to subsample edges on Algorithms 7.1, Line 11 and 7.1, Line 26 for inputs $\mathcal{S}, \mathcal{S}'$ respectively. We proceed under the coupling that the coins used to process common edges $e_j = e'_j$ for $j \leq t^*$ are identical. The key structural fact is that, under this coupling, the current sample sizes differ by at most 2 at every time. Indeed, for every common edge, the same Bernoulli coins are used in both executions, so all common edges are sampled and later resampled identically whenever the public transcript prefix is the same. Thus the only discrepancy comes from the extra update $e_{t^*} = \{u, v\}$. If $t < t^*$, then the sample sets are identical, $S_t = S'_t$. Assume now that $t \geq t^*$. Processing e_{t^*} can contribute at most one additional sampled edge, namely e_{t^*} itself. Moreover, it can cause premature deletion of sampled edges only through the counters at its endpoints u and v. For a fixed endpoint $w \in \{u, v\}$, the sampled edges incident to w always have distinct counter values, because counters are incremented only when a later adjacent edge arrives. Hence at any moment there is at most one incident sampled edge whose counter can cross the threshold $\tilde{\alpha}$ one step earlier because of the presence of e_{t^*}. So the discrepancy created at endpoint w contributes at most one additional missing edge. Summing over the two endpoints, we obtain

$$\big||S_t| - |S'_t|\big| \leq 2 \qquad \text{for every } t \in [T].$$

In summary, every query value $|S_t|$ used by either SVT instance has sensitivity at most 2 under an appropriate coupling.

The released sequence of estimates is a post-processing of the full SVT transcript, so it suffices to prove privacy for the SVT transcript. Let $a_t \sim A_t, b_t \sim B_t$ and $(r, r') \sim (R, R')$ where (R, R') denotes the coupled random coins above. We analyze privacy through the privacy loss variable

$$\mathcal{L} \coloneqq \log\left(\frac{\Pr[A_{1:T} = a_{1:T}, B_{1:T} = b_{1:T} \mid R = r]}{\Pr[A'_{1:T} = a_{1:T}, B'_{1:T} = b_{1:T} \mid R' = r']}\right).$$

Recall that it is enough to show $|\mathcal{L}| \leq \varepsilon$ with probability 1, since this is equivalent to pure ε-DP (Dwork and Roth, 2014, Lemma 3.17). By the chain rule of probability,

$$\mathcal{L} = \sum_{t=1}^{T} \log\left(\frac{\Pr[A_t = a_t \mid A_{<t} = a_{<t}, B_{<t} = b_{<t}, R = r]}{\Pr[A'_t = a'_t \mid A'_{<t} = a'_{<t}, B'_{<t} = b'_{<t}, R' = r']}\right) + \log\left(\frac{\Pr[B_t = b_t \mid A_{\leq t} = a_{\leq t}, B_{<t} = b_{<t}, R = r]}{\Pr[B'_t = b'_t \mid A'_{\leq t} = a'_{\leq t}, B'_{<t} = b'_{<t}, R' = r']}\right).$$

EstimateSVT **contribution.** Condition on $(A_{<t} = a_{<t}, B_{<t} = b_{<t}, R = r, R' = r')$. The state variables p_t and j_t are fixed and identical on the two neighboring streams. Hence the entire collection of threshold queries made by EstimateSVT at time t is obtained by repeatedly querying the same sensitivity-2 quantity $|S_t|$ against the public thresholds $p_t(1+\eta)^j$. By the standard privacy guarantee of SVT (Theorem A.7), the interactive transcript over all times is $(\varepsilon/2)$-DP.

SubsampleSVT **contribution.** Next condition on $(A_{\leq t} = a_{\leq t}, B_{<t} = b_{<t}, R = r, R' = r')$. At this point, the sampled set S_t is already determined, and the query made to SubsampleSVT is exactly the sensitivity-2 quantity $|S_t|$ compared against the fixed public threshold $\frac{a_3 \log^2(n)}{\varepsilon\eta^2}$. Again, the standard privacy guarantee of SVT ensures that the interactive transcript over all times is $(\varepsilon/2)$-DP.

Combining the two bounds concludes the proof. □

E.2. **Approximation and Space Guarantees.** We now show the approximation guarantees of our algorithm. The proofs below modify the proofs of McGregor and Vorotnikova (2018) in the non-private setting to the private setting. In particular, we modify their proof to hold for any $\tilde{\alpha} \geq \alpha$.

Let $M(G_t)$ be the maximum size of a matching in input graph G_t (consisting of all updates up to and including update e_t). We wish to compute an approximation of $M(G_t)$. Let B_u^t be the *last* $\tilde{\alpha}+1$ edges incident to $u \in V$ that appear in G_t. Let $E_{\tilde{\alpha}}^t$ be the set of edges $\{u,v\} \in G_t$ where $|B_u^t| \leq \tilde{\alpha}$ and $|B_v^t| \leq \tilde{\alpha}$. That is, $E_{\tilde{\alpha}}^t$ consists of the set of edges $\{u,v\}$ where the number of edges incident to u and v that appear in the stream after $\{u,v\}$ is at most $\tilde{\alpha}$. We say an edge $\{u,v\}$ is ***good*** if $\{u,v\} \in B_u^t \cap B_v^t$, and an edge is ***wasted*** if $\{u,v\} \in B_u^t \oplus B_v^t = (B_u^t \cup B_v^t) \setminus (B_u^t \cap B_v^t)$. Then, $E_{\tilde{\alpha}}^t$ is precisely the set of good edges in G^t. In other words, $E_{\tilde{\alpha}}^t = \bigcup_{u \neq v \in V} \left(B_u^t \cap B_v^t\right)$.

Our algorithm estimates the cardinality of $E_{\tilde{\alpha}}^t$ as an approximation of $M(G_t)$. The following lemma shows that $|E_{\tilde{\alpha}}^t|$, the number of good edges, serves as a proxy for the maximum matching size $M(G_t)$. Specifically, it is always at least as large as the maximum matching (since a matching can be found among the good edges) and at most an $(\tilde{\alpha}+2)$ factor larger (via a fractional matching argument using Edmonds' polytope theorem). This sandwich inequality is the foundation of our approximation guarantee, as it reduces the problem of estimating the matching size to the problem of estimating $|E_{\tilde{\alpha}}^t|$.

Lemma E.3. $M(G^t) \leq |E_{\tilde{\alpha}}^t| \leq (\tilde{\alpha}+2) \cdot M(G_t)$.

Proof of Lemma E.3. We first prove the right-hand side of this expression. To do this, we define a fractional matching using $E^t_{\tilde{\alpha}}$. Let $Y_e = \frac{1}{\tilde{\alpha}+1}$ if $e \in E^t_{\tilde{\alpha}}$ and $Y_e = 0$ otherwise. Then, $\{Y_e\}_{e\in E}$ is a fractional matching with maximum weight $\frac{1}{\tilde{\alpha}+1}$.[14] We now show a corollary of Edmonds' matching polytope theorem (Edmonds, 1965). Edmonds' matching polytope theorem implies that if the weight of a fractional matching on any induced subgraph $S \subseteq G$ is at most $\frac{|S|-1}{2}$, then the weight on the entire graph is at most $M(G)$. Now, we show the following proposition:

Proposition E.4. *Let $\{Y_e\}_{e\in E}$ be a fractional matching where the maximum weight on any edge is ψ. Then, $\sum_{e\in E} Y_e \leq (1+\psi)\cdot M(G)$.*

Proof of Proposition E.4. Let S be an arbitrary subset of vertices, and let $E(S)$ be the edges in the induced subgraph of S. We know that $|E(S)| \leq \frac{|S|(|S|-1)}{2}$ and by the definition of fractional matching, $\sum_{e\in E(S)} Y_e = \frac{\sum_{u\in S}\sum_{v\in N(u)} Y_{\{u,v\}}}{2} \leq \frac{\sum_{u\in S} 1}{2} = \frac{|S|}{2}$ where $\sum_{v\in N(u)} Y_{\{u,v\}} \leq 1$ by the constraints of the fractional matching. Thus, we know that (given the maximum weight on any edge is ψ)

$$\sum_{e\in E(S)} Y_e \leq \min\left(\frac{|S|}{2}, \frac{\psi|S|(|S|-1)}{2}\right) \tag{E.1}$$

$$\leq \frac{|S|-1}{2}\cdot \min\left(\frac{|S|}{|S|-1}, \psi|S|\right) \tag{E.2}$$

$$\leq \frac{|S|-1}{2}\cdot(1+\psi). \tag{E.3}$$

We can show the last inequality by considering two cases:

- If $\frac{|S|}{|S|-1} \leq \psi|S|$, then

$$\frac{1}{|S|-1} \leq \psi \tag{E.4}$$

$$1+\frac{1}{|S|-1} \leq 1+\psi \tag{E.5}$$

$$\frac{|S|}{|S|-1} \leq 1+\psi. \tag{E.6}$$

- If $\psi|S| \leq \frac{|S|}{|S|-1}$, then

$$\psi \leq \frac{1}{|S|-1} \tag{E.7}$$

$$\psi|S| - \psi \leq 1 \tag{E.8}$$

$$\psi|S| \leq 1+\psi. \tag{E.9}$$

Finally, let $Z_e = \frac{Y_e}{1+\psi}$. We can use Edmonds' polytope theorem to show that $\sum_{e\in E} Z_e \leq \frac{|S|-1}{2} \leq M(G)$ and so $\sum_{e\in E} Y_e \leq (1+\psi)\sum_{e\in E} Z_e \leq (1+\psi)\cdot M(G)$. We have proven our corollary. □

[14] A fractional matching is defined to be a set of weights $f(e) \geq 0$ on every edge e where $\forall v \in V : \sum_{e\ni v} f(e) \leq 1$.

Using the proposition above with maximum weight $\frac{1}{\tilde{\alpha}+1}$ implies that $\sum_{e\in E} Y_e = \frac{|E^t_{\tilde{\alpha}}|}{\tilde{\alpha}+1} \leq (1+\frac{1}{\tilde{\alpha}+1})M(G) = \frac{\tilde{\alpha}+2}{\tilde{\alpha}+1}\cdot M(G)$. Hence, we have that $|E^t_{\tilde{\alpha}}| \leq (\tilde{\alpha}+2)\cdot M(G)$.

Now, we prove the left inequality. To prove the left inequality, let H be the set of vertices in G_t with degree at least $\tilde{\alpha}+1$. These are the ***heavy*** vertices. We also define the following variables:

- $w :=$ the number of good edges with *no* endpoints in H,
- $x :=$ the number of good edges with *exactly one* endpoint in H,
- $y :=$ the number of good edges with *two* endpoints in H,
- $z :=$ the number of *wasted* edges with *two* endpoints in H.

First, $|E^t_{\tilde{\alpha}}| = w+x+y$. Below, we omit the superscript t for clarity but everything holds with respect to t. Now, we show the following additional equalities. We will first calculate the number of edges in the B_u of every $u \in H$ in terms of the variables we defined above. Since every vertex $u \in H$ is heavy, $|B_u| = \tilde{\alpha}+1$. Hence, $\sum_{u\in H}|B_u| = (\tilde{\alpha}+1)|H|$. Every edge $\{u,v\}$ in each of these B_u must either be a good edge or a wasted edge by definition since $\{u,v\} \in B_u \cup B_v$. For each edge counted in x, it has one endpoint in H so it is counted in exactly one B_u in H. Furthermore, for each edge counted in z, it is also counted in the B_u of exactly one of its two endpoints since it is wasted (i.e. the other endpoint doesn't have at most $\tilde{\alpha}$ edges that come after it). This leaves every good edge counted in y which is counted in exactly two B_u's. Hence, $\sum_{u\in H}|B_u| = x+2y+z = (\tilde{\alpha}+1)|H|$.

Now, we can also compute $z+y \leq \alpha|H|$ since $z+y$ is at most the total number of edges in the induced subgraph consisting of H. Since we know that the graph has arboricity α, we also know (by the properties of arboricity) that the induced subgraph consisting of H has at most $\alpha|H|$ edges. Hence, we can sum our inequalities: $x+2y+z = (\tilde{\alpha}+1)|H|$ and $-z-y \geq -\alpha|H|$ to obtain $x+y \geq (\tilde{\alpha}-\alpha+1)|H|$. Finally, let E_L be the set of edges with no endpoints in H. Every edge $\{u,v\} \in E_L$ is good since $B_u = N(u)$ and $B_v = N(v)$ by assumption. Note that $w = |E_L|$. Therefore, $|E^t_{\tilde{\alpha}}| = w+x+y \geq |H|+|E_L|$ since $\tilde{\alpha} \geq \alpha$. We can show that $|H|+|E_L| \geq M(G)$ since we can partition the set of edges in the maximum matching into edges in E_L and edges incident to H. All of the edges in E_L can be in the matching and at most one edge incident to each vertex in H is in the matching. We have successfully proven our lower bound: $|E^t_{\tilde{\alpha}}| \geq M(G)$. □

We now have the following algorithm for estimating $|E^t_{\tilde{\alpha}}|$ for every $t \in [T]$. For every sampled edge, $e = \{u,v\}$, the algorithm also stores counters c^u_e and c^v_e for the degrees of u and v in the rest of the stream. This requires an additional factor of $O(\log(\tilde{\alpha}))$ space. Thus, the algorithm maintains the invariant that each edge stored in the sample is a good edge with respect to the current G_t.

Let $E^*_t = \max_{t'\leq t}(|E^{t'}_{\tilde{\alpha}}|)$. We now show that we get a $\left(1+\eta, \frac{\log^2(n)}{\varepsilon\eta}\right)$-approximation of E^*_t with high probability. First, we note that E^*_t satisfies $M(G^t) \leq E^*_t \leq (2+\tilde{\alpha})M(G_t)$ since $E^*_t \geq |E^t_{\tilde{\alpha}}|$, $M(G_{t'}) \leq M(G_t)$, and Lemma E.3.

We now state our main lemma for our algorithm. It shows that the sub-sampled estimate $|S_t|/p_t$ tracks the running maximum E^*_t of good edges up to a $(1+\eta)$ multiplicative factor and an $O(\varepsilon^{-1}\eta^{-1}\log^2 n)$ additive error, with high probability. The proof addresses two subtleties: (1) the adaptive nature of the sampling probability p_t, which is determined online by the algorithm rather than set a priori, and (2) the SVT noise, which must be absorbed into the approximation error without exceeding its query budget.

Lemma E.5. *Algorithm 7.1 returns a* $\left(1+\eta, \frac{\log^2(n)}{\varepsilon\eta}\right)$*-approximation of* E_t^* *for every* $t \in [T]$ *with probability at least* $1 - 1/\operatorname{poly}(n)$.

Proof. Fix some $t \in [T]$. We need to show that at time t, we do not exceed the SVT "above" budgets as well as that sub-sampling yields sufficiently concentrated estimates with probability $1 - 1/\operatorname{poly}(n)$. Taking a union bound over $T = \operatorname{poly}(n)$ timestamps then yields the desired result.

First, let $\tau = \frac{d\log^2(n)}{\varepsilon\eta^2}$ for some large enough constant $d > 0$ and define level i_t (starting with $i_t = 2$) to be the level that contains E_t^* if $2^{i_t-1} \cdot \tau < E_t^* \le 2^{i_t} \cdot \tau$. Level $i_t = 1$ is defined as $0 \le E_t^* \le 2 \cdot \tau$. Let p_t denote the marginal probability that an edge is sampled from $E_{\tilde{\alpha}}^t$. Note that this probability is prior to sub-sampling on Algorithm 7.1, Line 23. In a perfect situation, at time t, edge e is sampled in level i with probability $p_t' := 2^{-i_t}$ when $i_t \ge 2$ and with probability $p_t' = 1$ when $i_t = 1$, but this is not the case as our algorithm determines p_t adaptively depending on the size of the sampled edges at time t.

SVT Budget. We show that with probability at least $1 - 1/\operatorname{poly}(n)$, neither of the SVT "above" budgets Q_1, Q_2 in Algorithm 7.1 is exceeded. First, for the sub-sampling SVT, we take $T = \operatorname{poly}(n)$ noisy comparisons with $\mathsf{Lap}(O(\varepsilon/\log(n)))$ noise. Hence with probability $1 - 1/\operatorname{poly}(n)$, all Laplace realizations are of order $O(\varepsilon^{-1}\log^2(n))$.

$Q_1 = a_1 \log(n)$ bounds the number of times the sub-sampling SVT in Algorithm 7.1, Line 23 is satisfied. Each time this SVT is satisfied, we must have

$$|S_t| \ge a_3\varepsilon^{-1}\eta^{-2}\log^2(n) - O(\varepsilon^{-1}\log^2(n)) \ge \frac{a_3}{2}\varepsilon^{-1}\log^2(n) \tag{E.10}$$

for sufficiently large constant $a_3 > 0$, leading to the probability of sampling edges being halved. The expected number of edges that are sampled at any point is $p_t \cdot |E_{\tilde{\alpha}}^t| \le p_t \cdot n^2$. Suppose a_1 is sufficiently large and we halved the sampling probability $Q_1/2 \ge \lceil \log(n^2) \rceil$ times so that $p_t \le \frac{1}{n^2}$. We can upper bound the probability that the SVT answers "above" by considering the experiment where we obtain a sample $\tilde{S}_t$ by sampling each edge with probability $\tilde{p} = \frac{1}{|E_{\tilde{\alpha}}^t|} \ge p_t$. By a multiplicative Chernoff bound (Theorem A.1), for $\psi = \frac{a_3}{2}\varepsilon^{-1}\log^2(n) - 1$,

$$\begin{aligned}
\mathbf{P}\left[|S_t| \ge \frac{a_3}{2}\varepsilon^{-1}\log^2(n)\right] &\le \mathbf{P}\left[|\tilde{S}_t| \ge \frac{a_3}{2}\varepsilon^{-1}\log^2(n)\right] && \text{(E.11)}\\
&= \mathbf{P}[|\tilde{S}_t| \ge (1+\psi)\cdot 1] && \text{(E.12)}\\
&= \mathbf{P}[|\tilde{S}_t| \ge (1+\psi)\tilde{p}\cdot |E_{\tilde{\alpha}}^t|] && \text{(E.13)}\\
&\le \exp\left(-\frac{\psi^2\cdot 1}{2+\psi}\right) && \text{(E.14)}\\
&\le \exp\left(-\Omega(\log^2(n))\right) \qquad n \text{ sufficiently large} && \text{(E.15)}\\
&\le \frac{1}{\operatorname{poly}(n)}. && \text{(E.16)}
\end{aligned}$$

Hence, for large enough constants a_1, a_3, we do not exceed the subsampling SVT "above" budget.

A similar argument shows that we do not exceed the estimate SVT "above" budget of Q_2 with probability at least $1 - 1/\operatorname{poly}(n)$. In the worst case, the number of times p is halved is Q_1. Thus, the maximum value of $|S|/p$ is $2^{Q_1} \cdot n^2$ since the maximum number of edges in the

graph is upper bounded by n^2. Since we are increasing our threshold by a factor of $(1+\eta)$ each time, our threshold exceeds $2^{Q_1}n^2$ after $\log_{1+\eta}(2^{Q_1}n^2) = O(\eta^{-1}Q_1 + \eta^{-1}\log(n)) = O(\eta^{-1}Q_1)$ times. If we set a_2 to be a large enough constant, we do not exceed the "above" budget for the estimate SVT.

Concentration. In a perfect situation, edge e is sampled at time t with probability $p'_t = 2^{-i_t}$ for $i_t \geq 2$ and with probability $p'_t = 1$ for $i_t = 1$. In that case, we can use the multiplicative Chernoff bound to bound the probability that our estimate concentrates as our sampling probability would be sufficiently high to do so. However, it is not the case that p_t is guaranteed to be 2^{-i_t} since it is determined adaptively by Algorithm 7.1. Hence, we need a slightly more sophisticated analysis than simply using the multiplicative Chernoff bound with p'_t.

We consider two cases. If $p_t \geq p'_t$, then we can lower bound the probability of success by what we would obtain using p'_t and we can use the multiplicative Chernoff bound in this case. Now, suppose that $p_t < p'_t$; then, we claim that this case never occurs in Algorithm 7.1 with probability at least $1 - 1/\operatorname{poly}(n)$. We now formally present these arguments.

If $p_t \geq p'_t$, then we use a multiplicative Chernoff bound (Theorem A.1) to bound our probability of success. Let s_t be the number of edges we sampled for $E^t_{\tilde{\alpha}}$ prior to any sub-sampling. That is, we obtain s_t by sampling with probability p_t before Algorithm 7.1, Line 23. For $\psi := \eta \cdot E^*_t / |E^t_{\tilde{\alpha}}|$,

$$\mathbf{P}\left[s_t - p_t \cdot |E^t_{\tilde{\alpha}}| \geq \eta \cdot p_t \cdot E^*_t\right] = \mathbf{P}\left[s_t - p_t \cdot |E^t_{\tilde{\alpha}}| \geq \psi \cdot p_t \cdot |E^t_{\tilde{\alpha}}|\right] \tag{E.17}$$

$$\leq \exp\left(-\frac{\psi^2}{2+\psi} \cdot p_t \cdot |E^t_{\tilde{\alpha}}|\right) \tag{E.18}$$

$$\leq \exp\left(-\frac{\psi^2}{2+\psi} \cdot p'_t \cdot |E^t_{\tilde{\alpha}}|\right). \qquad p_t \geq p'_t \tag{E.19}$$

When $\psi \geq 1$, we have $\frac{\psi}{2+\psi} \geq \frac{1}{3}$ and this is at most

$$\exp\left(-\frac{\psi}{3} \cdot p'_t \cdot |E^t_{\tilde{\alpha}}|\right) = \exp\left(-\frac{\eta E^*_t}{3} p'_t\right) \tag{E.20}$$

$$\leq \exp\left(-\Omega(\varepsilon^{-1}\eta^{-1}\log^2(n))\right) \quad \text{definition of } p'_t, \text{ sufficiently large } a_3 \tag{E.21}$$

$$= \frac{1}{\operatorname{poly}(n)}. \tag{E.22}$$

When $\psi \in (0,1)$, this is at most

$$\exp\left(-\frac{\psi^2}{3} \cdot p'_t \cdot E^*_t\right) \leq \exp\left(-\eta^2 E^*_t p'_t\right) \tag{E.23}$$

$$\leq \exp\left(-\Omega(\varepsilon^{-1}\log^2(n))\right). \quad \text{definition of } p'_t, \text{ sufficiently large } a_3 \tag{E.24}$$

$$= \frac{1}{\operatorname{poly}(n)} \tag{E.25}$$

Since we need only consider the case of $\psi \in (0,1)$ for the lower bound, an identical calculation yields the same concentration bound above but for the lower bound. All in all, $\frac{s_t}{p_t} \in [|E^t_{\tilde{\alpha}}| \pm \eta E^*_t]$ with high probability.

Now, we consider the case when $p_t < p'_t$. We show that with probability at least $1 - 1/\operatorname{poly}(n)$, this case *does not occur*. We prove this via induction on t. For $t = 1$, $p_t = p'_t$ trivially since both equal 1. Now, we assume our claim holds for t and show it holds for $t+1$. First, note that E^*_t cannot decrease so p'_t cannot increase as t increases. Then, by Equation (E.10), $p_{t+1} < p'_{t+1}$ only if we sample more than $\frac{a'_3 \log^2(n)}{2\varepsilon\eta^2}$ elements of $E^t_{\tilde{\alpha}}$ at probability $p_t = p'_t$ before Algorithm 7.1, Line 23. However, we have just shown that with high probability,

$$s_t \le (1+\eta)p_t E^*_t \le 2p'_t E^*_t \le 2\tau < \frac{a_3 \log^2(n)}{2\varepsilon\eta^2} \tag{E.26}$$

by the definition of p'_t, τ, and assuming a sufficiently large a_3. Hence this event does not occur with high probability.

Finally, the estimate SVT ensures that with high probability,

$$s_t \le p_t(1+\eta)^{j_t} + O(\varepsilon^{-1}\eta^{-1}\log^2(n)) \tag{E.27}$$

$$s_t > p_t(1+\eta)^{j_t - 1} - O(\varepsilon^{-1}\eta^{-1}\log^2(n)). \tag{E.28}$$

But we wish to approximate s_t/p_t, and dividing the inequalities by p_t yields

$$\frac{s_t}{p_t} \le (1+\eta)^{j_t} + \frac{O(\varepsilon^{-1}\eta^{-1}\log^2(n))}{p_t} \tag{E.29}$$

$$\frac{s_t}{p_t} > (1+\eta)^{j_t - 1} - \frac{O(\varepsilon^{-1}\eta^{-1}\log^2(n))}{p_t}. \tag{E.30}$$

If $i_t = 1$ then $p_t = 1$ and we are done. Thus consider the case where $i_t \ge 2$. We have shown above that $p_t \ge p'_t = 2^{-i_t}$ and $2^{i_t - 1}\tau \le E^*_t$ for $i_t \ge 2$. Thus in this case,

$$\frac{O(\varepsilon^{-1}\eta^{-1}\log^2(n))}{p_t} \le \frac{O(\varepsilon^{-1}\eta^{-1}\log^2(n))E^*_t}{\tau} \tag{E.31}$$

$$= O(\varepsilon^{-1}\eta^{-1}\log^2(n))E^*_t \cdot \frac{\varepsilon\eta^2}{d\log^2(n)} \tag{E.32}$$

$$\le \eta E^*_t \tag{E.33}$$

assuming a sufficiently large constant d.

Thus, combining the case work above with our concentration bounds on s_t/p_t, we see that $(1+\eta)^{j_t}$ is a $\left(1+\eta, O\left(\varepsilon^{-1}\eta^{-1}\log^2 n\right)\right)$-approximation of E^*_t.

□

The following lemma establishes the space complexity of our algorithm. Since the SVT mechanism controls the size of the sample S and triggers sub-sampling whenever $|S|$ grows too large, the number of stored edges remains bounded with high probability. Each stored edge carries two counters of size $O(\log \tilde{\alpha})$, yielding a truly sublinear space bound.

Lemma E.6. *Algorithm 7.1 requires $O\left(\frac{\log^2(n)\log(\tilde{\alpha})}{\varepsilon\eta^2}\right)$ space, with high probability.*

Proof. In terms of space complexity, we need $O\left(\frac{\log^2(n)\log(\tilde{\alpha})}{\varepsilon\eta^2}\right)$ space to store the sampled edges and the two counters per edge. We check when the number of sampled edges exceeds $\frac{a_3\log^2(n)}{\varepsilon\eta^2}$ in Algorithm 7.1, Line 24 using an SVT check. By the guarantees of SVT, the number of sampled edges is at most $\frac{a_3\log^2(n)}{\varepsilon\eta^2} + \frac{c'\log(n)}{\varepsilon} = O\left(\frac{\log^2(n)}{\varepsilon\eta^2}\right)$ for some large enough

constant $c' > 0$, with high probability, before it exceeds the threshold. Thus, with high probability, we store $O\left(\frac{\log^2(n)}{\varepsilon\eta^2}\right)$ sampled edges and each edge uses $O(\log(\tilde{\alpha}))$ bits to store the counters, resulting in our stated space bounds. □

Now, combining Lemmas E.3, E.5 and E.6 gives us our final accuracy guarantee in Theorem 7.1.

Appendix F. Proofs of Theorems 8.1 to 8.3

We now fill in the omitted details from Section 8.

F.1. **Inner Product Queries.** Given a private database $y \in \{0,1\}^n$ and a set of k linear queries $q^{(j)} \in \{0,1\}^n, j \in [k]$, the *inner product problem* asks for the values

$$\langle y, q^{(j)} \rangle \in \mathbb{Z}_{\geq 0}\,, \qquad \forall j \in [k]\,.$$

We use the following lower bound stated by Jain et al. (2023a) that is based on the works of Dinur and Nissim (2003); Dwork et al. (2007); Mir et al. (2011); De (2012).

Theorem F.1 (Theorem 4.5 in (Jain et al., 2023a))**.** *There are constants $c_1, c_2 > 0$ such that, for sufficiently large $n > 0$: if an algorithm $\mathcal{A}$ answers $c_1 n$ inner product queries within additive error $c_2\sqrt{n}$ with probability at least 0.99, then $\mathcal{A}$ is not $(1, 1/3)$-DP.*

F.2. **Maximum Cardinality Matching.** We now restate and prove the lower bound for computing the size of a maximum matching in the fully-dynamic continual release model.

Theorem F.2. *Fix $\varepsilon \in (0,1)$. If $\mathcal{A}$ is an ε-DP mechanism that answers maximum matching queries on graphs with n vertices in the continual release model within additive error ζ with probability at least 0.99 for fully dynamic streams of length T, then*

$$\zeta = \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right).$$

Moreover, we may assume the graph is bipartite, has maximum degree 2, and arboricity 1.

F.2.1. *Description of Reduction.* We reduce to the inner product query problem similar to Jain et al. (2023a). Let n be the dimension of a secret dataset $y \in \{0,1\}^n$. Consider $3n$ vertices $V = \{u_i, v_i, w_i : i \in [n]\}$. We use the first $\Theta(n)$ updates to encode the non-zero bits of y as a matching between the vertices u_i, v_i, i.e. we add the edge $\{u_i, v_i\}$ if and only if $y_i = 1$. Note that the size of the maximum matching is precisely $\|y\|_0$ by construction.

Given linear queries $q^{(1)} \in \{0,1\}^n$, we make the following updates: For each i such that $q_i^{(1)} = 1$, add the edge $\{v_i, w_i\}$. Then the size of the maximum matching is $\|y \mid q^{(1)}\|_0$, where $a \mid b$ denotes the bitwise OR of $a, b \in \{0,1\}^n$. The inner product can then be recovered using the inclusion-exclusion principle:

$$\langle y, q^{(1)} \rangle = \|y\|_0 + \|q^{(1)}\|_0 - \|y \mid q^{(1)}\|_0.$$

We can then delete the added edges and process the next query $q^{(2)}$ and so on.

Since emulating each query requires $\Theta(n)$ updates, we can answer $\Theta(n)$ inner product queries after $T = \Theta(n^2)$ updates. At a high level, since the error lower bound for $O(n)$ inner product queries is $\Omega(\sqrt{n})$, we have a lower bound of $\Omega(\min(n^{1/2}, T^{1/4}))$.

Algorithm F.1: Reduction from Inner Product Queries to Maximum Matching

Function `Alg`(*private dataset* $y \in \{0,1\}^n$*, public queries* $q^{(1)}, \dots, q^{(k)} \in \{0,1\}^n$*, DP mechanism for matchings* $\mathcal{M}$)
- $V \leftarrow \{u_i, v_i, w_i : i \in [n]\}$ ($3n$ vertices)
- $E_0 \leftarrow \varnothing$ empty edge set
- $S^{(0)} \leftarrow$ empty update stream of length n
- $S^{(1)}, \dots, S^{(k)} \leftarrow k$ empty update streams each of length $2n$
- **for** $i = 1, 2, \dots, n$ **do**
  - **if** $y_i = 1$ **then**
    - $S_i^{(0)} \leftarrow +\{u_i, v_i\}$ (insert edge $\{u_i, v_i\}$)
- **for** $j = 1, 2, \dots, k$ **do**
  - **for** $i = 1, 2, \dots, n$ **do**
    - **if** $q_i^{(j)} = 1$ **then**
      - $S_i^{(j)} \leftarrow +\{v_i, w_i\}$
      - $S_{n+i}^{(j)} \leftarrow -\{v_i, w_i\}$ (delete edge $\{v_i, w_i\}$)
- $S \leftarrow S^{(0)} + S^{(1)} + \cdots + S^{(k)}$ concatenation of all streams
- $r^{(0)}, r^{(1)}, \dots, r^{(k)} \leftarrow \mathcal{M}(S)$ answers to queries
- **for** $j = 1, 2, \dots, k$ **do**
  - Output $\|q^{(j)}\|_0 + r_n^{(0)} - r_n^{(j)}$ as approximate answer to $\langle q^{(j)}, y \rangle$ ($\langle q^{(j)}, y \rangle = \|q^{(j)}\|_0 + \|y\|_0 - \|q^{(j)} \mid y\|_0$)

F.2.2. *Proof of Lower Bound.*

Lemma F.3. *Given an ε-DP mechanism for maximum matching in the continual release setting that outputs estimates within additive error at most ζ with probability 0.99 for a fully-dynamic stream of length $T \geq n + 2nk$, Algorithm F.1 is an ε-DP mechanism for answering k inner product queries within additive error 2ζ with probability 0.99.*

Proof of Lemma F.3. Let $\mathcal{M}$ be such an ε-DP mechanism and suppose we are given a private dataset $y \in \{0,1\}^n$ as well as public linear queries $q^{(1)}, \dots, q^{(k)}$. Let $S^{(0)}, S^{(1)}, \dots, S^{(k)}$ be the partial update streams as in Algorithm F.1 and

$$(r^{(0)}, r^{(1)}, \dots, r^{(k)}) = \mathcal{M}(S^{(0)} + S^{(1)} + \cdots + S^{(k)})$$

be the corresponding stream of outputs when we run $\mathcal{M}$ on the concatenation of partial streams. By assumption, $r_n^{(0)}$ is an estimate of $\|y\|_0$ within additive error ζ and $r_n^{(j)}$ is an estimate of $\|y \mid q^{(j)}\|_0$ within additive error ζ. Hence the value of

$$\|q^{(j)}\|_0 + r_n^{(0)} - r_n^{(j)}$$

is an estimate of $\langle y, q^{(j)} \rangle$ with additive error within 2ζ. □

Theorem F.4. *If $\mathcal{A}$ is a 1-DP mechanism that answers maximum matching queries on graphs with n vertices in the continual release model within additive error ζ with probability at least 0.99 for fully dynamic streams of length T, then*

$$\zeta = \Omega(\min(\sqrt{n}, T^{1/4})).$$

Moreover, we may assume the graph is bipartite, has maximum degree 2, and arboricity 1.

Proof of Theorem F.4. Let c_1, c_2 be the constants from Theorem F.1. Fix $n \in \mathbb{N}$ and suppose $T \geq n + 2c_1 n^2$. By Lemma F.3, $\mathcal{A}$ implies a 1-DP mechanism for answering $k = c_1 n$ inner product queries within additive error 2ζ with probability 0.99. But then by Theorem F.1, we must have $\zeta \geq c_2\sqrt{n}/2$.

Now consider the regime of $T < n + 2c_1 n^2$. The same argument holds for $\tilde{n} = \Theta(\sqrt{T})$ such that $\tilde{n} + 2c_1\tilde{n}^2 \leq T$ to yield a lower bound of $\zeta \geq c_2\sqrt{\tilde{n}} = \Omega(T^{1/4})$. □

Lemma F.5. *Let $\varepsilon \in (0,1)$, $\ell := \lfloor 1/\varepsilon \rfloor$, $\tilde{T} \geq \ell$, $\zeta : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be an increasing error function. Suppose there is an ε-DP mechanism $\mathcal{A}$ for answering maximum matching queries on graphs of $\tilde{n}$ vertices in the continual release model that outputs estimates within additive error at most*

$$\ell \cdot \zeta(\tilde{n}/\ell, \tilde{T}/\ell) = O\left(\frac{\zeta(\varepsilon\tilde{n}, \varepsilon\tilde{T})}{\varepsilon}\right)$$

with probability 0.99 for fully-dynamic streams of length $\tilde{T}$.

Let $T = \tilde{T}/\ell$, and $n = \tilde{n}/\ell$. There is a 1-DP mechanism for answering maximum matching queries on graphs of n vertices in the continual release model that outputs estimates within additive error at most $\zeta(n, T)$ with probability 0.99 for fully-dynamic streams of length T.

Proof of Lemma F.5. Let S be an edge update stream of length T from a graph on n vertices.

We construct an edge update stream of length $\tilde{T} = \ell T$ from a graph on $\tilde{n} = \ell n$ vertices as follows: Duplicate each vertex v in the graph ℓ times into $v_1, \ldots, v_\ell$. For each update $S_i \in \{+\{u,v\}, -\{u,v\}, \perp\}$ from S, define the length ℓ partial stream

$$S^{(i)} := \begin{cases} +\{u_1,v_1\}, +\{u_2,v_2\}, \ldots, +\{u_\ell,v_\ell\}, & S_i = +\{u,v\} \\ -\{u_1,v_1\}, -\{u_2,v_2\}, \ldots, -\{u_\ell,v_\ell\}, & S_i = -\{u,v\} \\ \perp, \perp, \ldots, \perp, & S_i = \perp . \end{cases}$$

Here u_i, v_i are the copies of the original vertices u, v. Then we take the new stream $\tilde{S} := S^{(1)} + \cdots + S^{(T)}$ to be the concatenation of all partial streams.

Let $\tilde{r} = \mathcal{A}(\tilde{S})$ be the result of the algorithm on $\tilde{S}$ and output $r = (\tilde{r}_\ell/\ell, \tilde{r}_{2\ell}/\ell, \ldots, \tilde{r}_{T\ell}/\ell)$ as the query answers to the original stream S.

By construction, the size of the maximum matching $\mu(\tilde{S}_{i\ell})$ in the new stream at time $i\ell, i \in [T]$ is exactly $\ell \cdot \mu(S_i)$ in the original stream at time i. Thus by assumption, our output has additive error within

$$\frac{\zeta(\varepsilon \cdot \ell n, \varepsilon \cdot \ell T)}{\varepsilon\ell} = O(\zeta(n, T))$$

with probability 0.99. Any neighboring streams from S become ℓ-neighboring streams from $\tilde{S}$ and thus by group privacy, our output is 1-DP. □

We are now ready to prove Theorem 8.1.

Proof of Theorem 8.1. In the regime $T \geq 1/\varepsilon$, combining Lemma F.5 and Theorem F.4 implies a lower bound of

$$\frac{1}{\varepsilon}\Omega(\min(\sqrt{\varepsilon n}, (\varepsilon T)^{1/4})) = \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}\right)\right).$$

In the second regime $T < 1/\varepsilon, n/2$, we prove a lower bound of $\Omega(T)$. Suppose towards a contradiction that there is an ε-DP mechanism in this regime with additive error within $\zeta = T/4$. Let $r, \tilde{r}$ be the outputs of $\mathcal{A}$ on the empty update stream of length T and the stream that adds an edge of a fixed perfect matching at every time step, respectively. By the accuracy of $\mathcal{A}$, we have $\mathbf{P}[r_T \leq T/4] \geq 0.99$. By ε-DP and group privacy, we have

$$\mathbf{P}[\tilde{r}_T > T/4] \leq e^{\varepsilon T}\mathbf{P}[r_T > T/4] \leq 0.01e < 0.99.$$

But then since the true matching size in the second stream at time T is T, $\mathcal{A}$ does not have additive error at most $\zeta = T/4$ with probability at least 0.99. By contradiction, it follows that we must have $\zeta = \Omega(T)$.

In the final regime $n/2 \leq T < 1/\varepsilon$, we show a lower bound of $\Omega(n)$. Simply apply our second argument with $\tilde{T} = n/2$ to arrive at a lower bound of $\Omega(\tilde{T}) = \Omega(n)$. □

F.3. **Triangle Counting.** We now prove Theorem 8.2, whose statement we repeat below for convenience.

Theorem F.6. *Fix $\varepsilon \in (0, 1)$. If $\mathcal{A}$ is an ε-DP mechanism that answers triangle counting queries on graphs with n vertices in the continual release model within additive error ζ with probability at least 0.99 for fully dynamic streams of length T, then*

$$\zeta = \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right).$$

Moreover, we may assume the graph has maximum degree 2 and arboricity 2.

F.3.1. *Description of Reduction.* Using a similar technique, we show a lower bound for the problem of privately estimating the number of triangles. In particular, we encode a single bit of a private database $y \in \{0, 1\}^n$ using a subgraph on 3 vertices u, v, w. The edge $\{u, v\}$ always exists but $\{v, w\}$ exists if and only if the bit is non-zero. Then notice that adding the edge $\{u, w\}$ creates a triangle if and only if the bit is non-zero. Thus after adding the edge $\{u_i, w_i\}$ for every i such that $q_i = 1$, the number of triangles in the graph is precisely $\langle y, q\rangle$, so no inclusion-exclusion step is required. Observe that at all times, the graph maintained by the reduction has maximum degree 2 and arboricity at most 2.

Algorithm F.2: Reduction from Inner Product Queries to Triangle Counting

Function `Alg`(*private dataset* $y \in \{0,1\}^n$, *public queries* $q^{(1)}, \dots, q^{(k)} \in \{0,1\}^n$, *DP mechanism for triangle counting* $\mathcal{M}$)
- $V \leftarrow \{u_i, v_i, w_i : i \in [n]\}$ ($3n$ vertices)
- $E_0 \leftarrow \varnothing$ empty edge set
- $S^{(0)} \leftarrow$ empty update stream of length $2n$
- $S^{(1)}, \dots, S^{(k)} \leftarrow k$ empty update streams each of length $2n$
- **for** $i = 1, 2, \dots, n$ **do**
  - $S_i^{(0)} \leftarrow +\{u_i, v_i\}$ (insert edge $\{u_i, v_i\}$)
  - **if** $y_i = 1$ **then**
    - $S_{n+i}^{(0)} \leftarrow +\{v_i, w_i\}$
- **for** $j = 1, 2, \dots, k$ **do**
  - **for** $i = 1, 2, \dots, n$ **do**
    - **if** $q_i^{(j)} = 1$ **then**
      - $S_i^{(j)} \leftarrow +\{u_i, w_i\}$
      - $S_{n+i}^{(j)} \leftarrow -\{u_i, w_i\}$ (delete edge $\{u_i, w_i\}$)
- $S \leftarrow S^{(0)} + S^{(1)} + \cdots + S^{(k)}$ concatenation of all streams
- $r^{(0)}, r^{(1)}, \dots, r^{(k)} \leftarrow \mathcal{M}(S)$ answers to queries
- **for** $j = 1, 2, \dots, k$ **do**
  - Output $r_n^{(j)}$ as approximate answer to $\langle q^{(j)}, y \rangle$ (the number of triangles after inserting the query edges is $\langle q^{(j)}, y \rangle$)

F.3.2. *Proof of Lower Bound.*

Lemma F.7. *Given an ε-DP mechanism for triangle counting in the continual release setting that outputs estimates within additive error at most ζ with probability 0.99 for a fully-dynamic stream of length $T \geq 2n + 2nk$, Algorithm F.2 is an ε-DP mechanism for answering k inner product queries within additive error ζ with probability 0.99.*

Proof of Lemma F.7. Let $\mathcal{M}$ be such an ε-DP mechanism and suppose we are given a private dataset $y \in \{0,1\}^n$ as well as public linear queries $q^{(1)}, \dots, q^{(k)}$. Let $S^{(0)}, S^{(1)}, \dots, S^{(k)}$ be the partial update streams as in Algorithm F.2 and

$$(r^{(0)}, r^{(1)}, \dots, r^{(k)}) = \mathcal{M}(S^{(0)} + S^{(1)} + \cdots + S^{(k)})$$

be the corresponding stream of outputs when we run $\mathcal{M}$ on the concatenation of partial streams. By construction, at the time of the n-th update within the j-th partial stream, the vertices u_i, v_i, w_i form a triangle if and only if $y_i = 1$ and $q_i^{(j)} = 1$, so that the number of triangles in the graph is precisely $\langle y, q^{(j)} \rangle$. By assumption, $r_n^{(j)}$ is an estimate of $\langle y, q^{(j)} \rangle$ within additive error ζ. □

Theorem F.8. *If $\mathcal{A}$ is a 1-DP mechanism that answers triangle counting queries on graphs with n vertices in the continual release model within additive error ζ with probability at least*

0.99 *for fully dynamic streams of length T, then*

$$\zeta = \Omega(\min(\sqrt{n}, T^{1/4})).$$

Moreover, we may assume the graph has maximum degree 2, and arboricity 2.

Proof of Theorem F.8. Let c_1, c_2 be the constants from Theorem F.1. Fix $n \in \mathbb{N}$ and suppose $T \geq 2n + 2c_1 n^2$. By Lemma F.7, $\mathcal{A}$ implies a 1-DP mechanism for answering $k = c_1 n$ inner product queries within additive error ζ with probability 0.99. But then by Theorem F.1, we must have $\zeta \geq c_2\sqrt{n}$.

Now consider the regime of $T < 2n + 2c_1 n^2$. The same argument holds for $\tilde{n} = \Theta(\sqrt{T})$ such that $2\tilde{n} + 2c_1\tilde{n}^2 \leq T$ to yield a lower bound of $\zeta \geq c_2\sqrt{\tilde{n}} = \Omega(T^{1/4})$. □

By following an identical argument to Lemma F.5, i.e. creating duplicate instances of the lower bound construction within a longer stream (note that the ℓ duplicated copies of the construction are vertex-disjoint, so the number of triangles is multiplied by exactly ℓ), we derive a similar lower bound to Theorem 8.1 that accounts for the privacy parameter ε.

F.4. **Connected Components.** We now prove Theorem 8.3, whose statement we repeat below for convenience.

Theorem F.9. *Fix $\varepsilon \in (0, 1)$. If $\mathcal{A}$ is an ε-DP mechanism that answers connected component queries on graphs with n vertices in the continual release model within additive error ζ with probability at least* 0.99 *for fully dynamic streams of length T, then*

$$\zeta = \Omega\left(\min\left(\sqrt{\frac{n}{\varepsilon}}, \frac{T^{1/4}}{\varepsilon^{3/4}}, n, T\right)\right).$$

Moreover, we may assume the graph is bipartite, has maximum degree 2, and arboricity 2.

F.4.1. *Description of Reduction.* Using a similar technique, we show a lower bound for the problem of privately estimating the number of connected components. In particular, we encode a single bit of a private database $y \in \{0, 1\}^n$ using a subgraph on 4 vertices u, v, a, b. The edges $\{u, a\}, \{v, b\}$ always exist but $\{a, b\}$ exists if and only if the bit is non-zero. Then notice that adding the edge $\{u, v\}$ decreases the number of connected components if and only if the bit is zero.

F.4.2. *Proof of Lower Bound.*

Lemma F.10. *Given an ε-DP mechanism for connected components in the continual release setting that outputs estimates within additive error at most ζ with probability* 0.99 *for a fully-dynamic stream of length $T \geq 3n + 2nk$, Algorithm F.3 is an ε-DP mechanism for answering k inner product queries within additive error 2ζ with probability* 0.99*.*

Proof of Lemma F.10. Let $\mathcal{M}$ be such an ε-DP mechanism and suppose we are given a private dataset $y \in \{0, 1\}^n$ as well as public linear queries $q^{(1)}, \ldots, q^{(k)}$. Let $S^{(0)}, S^{(1)}, \ldots, S^{(k)}$ be the partial update streams as in Algorithm F.3 and

$$(r^{(0)}, r^{(1)}, \ldots, r^{(k)}) = \mathcal{M}(S^{(0)} + S^{(1)} + \cdots + S^{(k)})$$

Algorithm F.3: Reduction from Inner Product Queries to Connected Components

Function `Alg`(*private dataset* $y \in \{0,1\}^n$, *public queries* $q^{(1)}, \dots, q^{(k)} \in \{0,1\}^n$, *DP mechanism for connected components* $\mathcal{M}$)
- $V \leftarrow \{u_i, v_i, a_i, b_i : i \in [n]\}$ ($4n$ vertices)
- $E_0 \leftarrow \varnothing$ empty edge set
- $S^{(0)} \leftarrow$ empty update stream of length $3n$
- $S^{(1)}, \dots, S^{(k)} \leftarrow k$ empty update streams each of length $2n$
- **for** $i = 1, 2, \dots, n$ **do**
  - $S_i^{(0)} \leftarrow +\{u_i, a_i\}$ (insert edge $\{u_i, a_i\}$)
  - $S_{n+i}^{(0)} \leftarrow +\{v_i, b_i\}$
  - **if** $y_i = 1$ **then**
    - $S_{2n+i}^{(0)} \leftarrow +\{a_i, b_i\}$
- **for** $j = 1, 2, \dots, k$ **do**
  - **for** $i = 1, 2, \dots, n$ **do**
    - **if** $q_i^{(j)} = 1$ **then**
      - $S_i^{(j)} \leftarrow +\{u_i, v_i\}$
      - $S_{n+i}^{(j)} \leftarrow -\{u_i, v_i\}$ (delete edge $\{u_i, v_i\}$)
- $S \leftarrow S^{(0)} + S^{(1)} + \cdots + S^{(k)}$ concatenation of all streams
- $r^{(0)}, r^{(1)}, \dots, r^{(k)} \leftarrow \mathcal{M}(S)$ answers to queries
- **for** $j = 1, 2, \dots, k$ **do**
  - Output $\|q^{(j)}\|_0 + r_n^{(j)} - r_{3n}^{(0)}$

be the corresponding stream of outputs when we run $\mathcal{M}$ on the concatenation of partial streams. By assumption, $(2n - r_{3n}^{(0)})$ is an estimate of $\|y\|_0$ within additive error ζ and $(2n - r_n^{(j)})$ is an estimate of $\|y \mid q^{(j)}\|_0$ within additive error ζ. Hence the value of

$$\|q^{(j)}\|_0 + (2n - r_{3n}^{(0)}) - (2n - r_n^{(j)}) = \|q^{(j)}\|_0 + r_n^{(j)} - r_{3n}^{(0)}$$

is an estimate of $\langle y, q^{(j)} \rangle$ with additive error within 2ζ. □

Theorem F.11. *If $\mathcal{A}$ is a 1-DP mechanism that answers connected component queries on graphs with n vertices in the continual release model within additive error ζ with probability at least 0.99 for fully dynamic streams of length T, then*

$$\zeta = \Omega(\min(\sqrt{n}, T^{1/4})).$$

Moreover, we may assume the graph is bipartite, has maximum degree 2, and arboricity 2.

Proof of Theorem F.11. Let c_1, c_2 be the constants from Theorem F.1. Fix $n \in \mathbb{N}$ and suppose $T \geq 3n + 2c_1 n^2$. By Lemma F.10, $\mathcal{A}$ implies a 1-DP mechanism for answering $k = c_1 n$ inner product queries within additive error 2ζ with probability 0.99. But then by Theorem F.1, we must have $\zeta \geq c_2\sqrt{n}/2$.

Now consider the regime of $T < 3n + 2c_1 n^2$. The same argument holds for $\tilde{n} = \Theta(\sqrt{T})$ such that $3\tilde{n} + 2c_1\tilde{n}^2 \leq T$ to yield a lower bound of $\zeta \geq c_2\sqrt{\tilde{n}} = \Omega(T^{1/4})$. □

By following an identical argument to Lemma F.5, i.e. creating duplicate instances of the lower bound construction within a longer stream, we derive a similar lower bound to Theorem 8.1 that accounts for the privacy parameter ε.

Appendix G. Private Static Densest Subgraph

Theorem G.1 (Theorem 6.1 in (Dhulipala et al., 2023)). *Fix $\eta \in (0, 1]$. There is an ε-edge DP densest subgraph algorithm that runs in polynomial time and $O(m+n)$ space and returns a subset $V' \subseteq V$ of vertices that induces a $\big(2, O(\varepsilon^{-1} \log n)\big)$-approximate densest subgraph with probability at least $1 - O(1/\operatorname{poly}(n))$.*

Theorem G.2 (Theorem 5.1 in (Dhulipala et al., 2022)). *Fix $\eta \in (0, 1]$. There is an ε-edge DP densest subgraph algorithm that runs in $O((m+n)\log^3 n)$ time and $O(m+n)$ space and returns a subset $V' \subseteq V$ of vertices that induces a $\big(1+\eta, O(\varepsilon^{-1}\eta^{-3}\log^4 n)\big)$-approximate densest subgraph with probability at least $1 - O(1/\operatorname{poly}(n))$ for any constant $\eta > 0$.*

Appendix H. The Density of the Densest Subgraph

We now recall the following theorem about computing the exact density of a densest subgraph in the static setting.

Theorem H.1. *Given an undirected graph $G = (V, E)$, there is an algorithm* ExactDensity *that computes the exact density of the densest subgraph. Moreover, the algorithm terminates in $O(|E|^3 \log(|V|))$ time and uses $O(|E|)$ space.*

We first describe the auxiliary flow graph described by Boob et al. (2020); Chekuri et al. (2022). Given an undirected graph $G = (V, E)$ and a value $\lambda > 0$, we construct the following bipartite flow network $D = (N, A)$. We use the terminology of nodes and arcs to distinguish from the original graph G. N consists of a source node s, a node a_e for every edge $e \in E$, a node a_v for every vertex $v \in V$, and a sink node t. There is a directed arc from s to a_e for every $e \in E$ with capacity 1, directed arcs from a_e to a_v and from a_e to a_u for every $e = \{u, v\} \in E$, both with infinite capacity, and a directed arc from a_v to t for every $v \in V$ with capacity λ.

For every $U \subseteq V$ in the original graph, the set of nodes

$$N_U := \{s\} \cup \{a_{uv} : u, v \in U\} \cup \{a_v : v \in U\}$$

is an (s, t)-cut with capacity

$$|E| - |E[U]| + \lambda|U|.$$

Any minimum (s, t)-cut must be of this form (allowing for $U = \varnothing$). This leads to the following observation.

Proposition H.2 ((Chekuri et al., 2022)). *Let $G = (V, E)$ be an undirected graph, $\lambda \geq |E|/|V|$, and $D = (N, A)$ be the directed flow graph obtained from G. Let N_U induce a minimum (s, t)-cut in D. Either the densest subgraph in G has density at most λ and $\delta(N_U)$ has capacity $|E|$, or $G[U]$ has density strictly greater than λ and $\delta(N_U)$ has capacity strictly less than $|E|$.*

Proof of Theorem H.2. For $U = \varnothing$, the cut induced by $N_\varnothing = \{s\}$ has capacity $|E| \leq \lambda|V|$.

Let $U \subseteq V$. If the density of $G[U]$ is at most λ, then $|E[U]| \leq \lambda|U|$ so that the cut induced by N_U has capacity at least $|E|$. If this holds for all $\varnothing \neq U \subseteq V$, then the cut induced by $N_\varnothing$ with capacity $|E|$ is a minimum cut and by the max-flow min-cut theorem, the maximum flow has value $|E|$.

Otherwise, assume that the density of $G[U^*]$ is strictly greater than λ. Then $|E[U^*]| > \lambda|U^*|$ so that the cut induced by N_{U^*} has capacity strictly less than $|E|$. Another application of the max-flow min-cut theorem concludes the proof. □

Now we recall the classical result of Dinitz (2006).

Theorem H.3 ((Dinitz, 2006))**.** *Given a directed graph $D = (N, A)$ with a source-sink pair $s, t \in N$ and non-negative arc capacities, there is an algorithm for computing an exact maximum (s, t)-flow. Moreover, the algorithm terminates in $O(|N|^2|A|)$ time while using $O(|N| + |A|)$ space.*

We are now ready to show Theorem H.1.

Proof of Theorem H.1. Without loss of generality, assume that G is connected so that $|E| = \Omega(|V|)$.

Construct the auxiliary flow network $D = (N, A)$ above for some guess $\lambda \geq {}^{|E|}/_{|V|}$ of the value of the densest subgraph. There are $2 + |V| + |E| = O(|E|)$ nodes and $|E| + 2|E| + |V| = O(|E|)$ arcs. We can check if the densest subgraph has value at most λ by computing the maximum flow in D using Dinitz's algorithm (Theorem H.3). This is justified by Theorem H.2. Moreover, each flow computation also retrieves the vertices inducing a minimum (s, t)-cut by computing the vertices reachable from s in the residual graph. The time and space complexity of computing the residual flow is dominated by the time and space complexity of the flow algorithm. Thus each flow computation either declares that the densest subgraph has density "at most λ", or produces some $U \subseteq V$ with density strictly more than λ.

The value of the densest subgraph is guaranteed to lie in the interval $[{}^{|E|}/_{|V|}, {}^{(|V|-1)}/_2] \subseteq [{}^1/_2, {}^{|V|}/_2]$, so we can binary search over λ in $O(\log {}^{|V|}/_\alpha)$ time and constant space to produce an estimate of the optimal density within an additive error of α. Note that the densest subgraph has density at least ${}^{|E|}/_{|V|}$.

We first use the flow gadget to check if the densest subgraph has density strictly greater than ${}^{|E|}/_{|V|}$. Thus at every iteration, we maintain a vertex set U and an upper bound $h \geq 0$ such that the density $\rho(U)$ of $G[U]$ and the optimal density ρ^* satisfy

$$\rho(U) \leq \rho^* \leq h.$$

Each iteration of the binary search takes time

$$O(|N|^2|A|) = O(|E|^3).$$

Thus the total running time is

$$O\left(|E|^3 \log({}^{|V|}/_\alpha)\right).$$

Here $\alpha > 0$ is the accepted level of additive error.

We remark that the density of a subgraph is a rational number whose numerator lies in $[|E|]$ and denominator in $[|V|]$. This means that the gap between the optimal density and the density of suboptimal subgraphs is at least ${}^1/_{|V|(|V|-1)} \geq {}^1/_{|V|^2}$. We can thus set

$\alpha = 1/|V|^2$ so that the density of the maintained subgraph U is optimal after termination. This brings the total running time to

$$O\left(|E|^3 \log(|V|^3)\right) = O\left(|E|^3 \log(|V|)\right).$$

Faster max-flow or min-cut algorithms lead to better running time.

The auxiliary space is proportional to the size of the digraph and is thus

$$O(|N| + |A|) = O(|E|).$$

□

Appendix I. Public Bound on Arboricity

Our edge-DP matching algorithm (Theorem 7.1) requires a public data-oblivious estimate $\tilde{\alpha}$ of the maximum arboricity α over the stream. While our algorithms *always* guarantee privacy, their utility is only guaranteed when $\tilde{\alpha} \geq \alpha$. One way to remove this assumption is to first execute an insertion-only continual release algorithm for estimating the arboricity and then run the desired algorithm for a second pass given the estimated arboricity.

I.1. **Notification of Failure.** In the one-pass setting, the simple approach above fails since we require a bound on the maximum arboricity throughout the entire stream. However, it may still be desirable to notify the observer that the current arboricity of the graph exceeds the public bound and thus the utility is no longer guaranteed. For this purpose, we can run a continual release algorithm for estimating the arboricity alongside the desired algorithm. For the sake of simplicity, we explain how we can use our k-core algorithm (Theorem 5.1) to accomplish such a task.

Recall that the *degeneracy* of a graph is defined to be the least number k such that every induced subgraph has a vertex of degree at most k. It is known that this quantity is equal to the value of the largest k-core, say denoted by $k_{\max}$. Using the definition of the arboricity α as the minimum number of forests that partition the edge set, we can show that

$$\frac{1}{2}k_{\max} \leq \alpha \leq k_{\max}.$$

Hence our k-core algorithm immediately yields a $(5, O(\varepsilon^{-1}\log^3(n)))$-approximation of the arboricity. When this estimated arboricity (approximately) exceeds the public bound, we can notify the observer that utility is no longer guaranteed by outputting "FAIL". Note that algorithms with better approximation and space guarantees that directly estimate α most likely exist, but we describe the approach here using Theorem 5.1 for the sake of simplicity.

I.2. **Guessing the Arboricity.** In the case of our matching algorithms (Section 7), we briefly sketch how to completely remove the dependence on a prior public $\tilde{\alpha}$ bound at the cost of additional space. Suppose we are given an ε-edge DP (β, ζ)-approximation continual release algorithm $\mathcal{A}_e$ for computing the arboricity that uses $O(S)$ space (e.g. see previous section). We can guess $\tilde{\alpha} = 2^p$ for each $p = 1, 2, \ldots, O(\log n)$ and execute $O(\log n)$ instances of our edge-DP continual release matching algorithm (Theorem 7.1) alongside $\mathcal{A}_e$. Thus we can identify the "correct" output from the parallel instances corresponding to the current arboricity α_t up to a $(2\beta, O(\zeta))$-approximation.

Plugging in our edge-DP matching algorithm (Theorem 7.1) and $\mathcal{A}_e$ yields an ε-edge DP continual release algorithm that returns a $\big(O(\beta\alpha_t + \zeta), O(\varepsilon^{-1}\operatorname{poly}\log(n))\big)$-approximate estimate of the maximum matching size using $O(S + \varepsilon^{-1}\operatorname{poly}\log(n))$ space.