Lumberjack: Better Differentially Private Random Forests through Heavy Hitter Detection in Trees

Christian Janos Lebeda[3,⋆,*], **David Erb**[2,⋆,†], **Tudor Cebere**[1], **Aurélien Bellet**[1]

[⋆]Lead authors

[1]PreMeDICaL, Inria, Université de Montpellier, INSERM, France
[2]Technical University of Munich, Germany
[3]University of Copenhagen, Denmark

Abstract

Random forests are widely used in fields involving sensitive tabular data, but existing approaches to enforcing differential privacy (DP) typically degrade performance to the point of impracticality. In this paper, we introduce `Lumberjack`, a differentially private random forest algorithm that achieves substantially higher utility by constructing large random decision trees and then applying aggressive, privacy-preserving pruning to retain only sufficiently populated nodes. A key component of our approach is a novel (ε, δ)-DP heavy hitter detection algorithm for hierarchical data, whose error is $O_{\varepsilon,\delta}(\sqrt{\log h})$ for trees of height h and may be of independent interest. This favorable scaling enables the use of significantly deeper trees than in prior work, leading to improved expressiveness under privacy constraints. Our empirical evaluation on benchmark datasets shows that `Lumberjack` consistently outperforms prior DP random forest methods, establishing a new state of the art. In particular, our approach yields substantial improvements in the privacy-utility trade-off for practical privacy budgets. Our findings suggest that carefully designed DP random forests can close much of the utility gap, highlighting a promising and underexplored direction for future research.

1 Introduction

Random forests (RFs) [Breiman, 2001, Geurts et al., 2006] remain among the most accurate and most widely used methods for tabular prediction, often matching or outperforming more complex deep learning approaches [Fernández-Delgado et al., 2014, Grinsztajn et al., 2022, Kaggle, 2022, Uddin and Lu, 2024]. Built as ensembles of decision trees that recursively partition the feature space and aggregate predictions across leaves, RFs combine strong predictive performance with several practical advantages: they naturally accommodate heterogeneous feature types, require comparatively little hyperparameter tuning, and are often substantially more computationally efficient than deep neural networks [Grinsztajn et al., 2022]. These properties have made them a central tool in many high-stakes applications involving sensitive tabular data, from healthcare to financial risk assessment.

At the same time, like other machine learning models, random forests can leak information about individual records used to train them. For instance, recent work has shown that, under certain conditions, some training examples can be reconstructed from publicly released random forest models [Ferry et al., 2024], motivating the need for formal privacy protections. Differential Privacy (DP) [Dwork et al., 2006] has emerged as the standard framework for addressing such risks by

[*]Work done as a postdoc at PreMeDICaL, Inria.
[†]Work done while visiting PreMeDICaL, Inria.

Preprint.

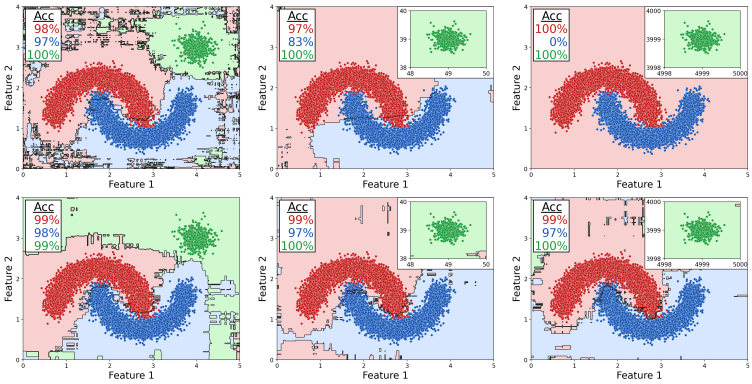

Figure 1: Toy example. Top row: fully random trees (depth=10) under ∞-DP. Bottom row: `Lumberjack` under $(2, 10^{-6})$-DP. As the outlier class boundary shifts and the feature-space range changes (left to right), fully random trees fail to capture the structure, whereas our method continues to construct accurate trees.

providing rigorous guarantees that an algorithm's output changes only minimally when any one individual's data is added or removed from the training set.

Much of the literature on differentially private machine learning has focused on differentiable models, particularly deep neural networks trained with private optimization methods such as DP-SGD [Abadi et al., 2016, Ponomareva et al., 2023]. However, this paradigm can incur substantial utility costs under privacy constraints, because private gradient-based training injects noise into a high-dimensional optimization process, with costs that scale unfavorably with model dimensionality [Tramèr and Boneh, 2021]. As a result, private deep models may underperform substantially simpler learning methods.

This motivates the exploration of alternative model classes with differently structured privacy-utility trade-offs, such as random forests. However, constructing differentially private RFs presents several challenges. The primary technical difficulty lies in building high-quality decision tree structures while satisfying strict privacy constraints. Decision tree algorithms recursively partition the dataset by selecting a feature and split threshold value. In standard non-private approaches [Breiman, 2001], these splits are chosen greedily by querying the data to optimize an impurity-based criterion. When adapted naively to the differentially private setting, such data-dependent procedures incur significant noise, leading to poor utility (e.g. [Patil and Singh, 2014, Fletcher and Islam, 2015a]).[3] A common workaround is to use fully random trees, inspired by extremely randomized trees [Geurts et al., 2006]. In these approaches [Jagannathan et al., 2012, 2013, Fletcher and Islam, 2015b, 2017, Holohan et al., 2019], tree structures are generated using splits that are completely independent of the data, while the data is used only to label leaves. Unfortunately, fully random trees also suffer from significant limitations. Their performance is highly sensitive to tree depth: shallow trees yield overly large leaves, while deep trees produce leaves with too few samples for reliable privatized predictions. They are also strongly dependent on the underlying data distribution and often fail to capture informative structure, collapsing to majority-class predictions. We illustrate this behavior on a toy example in Figure 1, using an infinite privacy budget to emphasize that this is an inherent limitation of data-independent tree construction. Consequently, DP ensembles of fully random trees typically perform poorly. Overall, existing differentially private random forest methods, including both greedy and fully random variants, are of limited practical utility.

Contributions. We introduce a novel differentially private random forest algorithm, `Lumberjack`. Our approach combines randomized tree construction, inspired by [Geurts et al., 2006] and prior work, with a principled privacy-aware mechanism that determines *when to stop splitting*. Concretely, we allocate part of the privacy budget to test whether a node contains a sufficient number of data points in feature space, allowing us to prune empty or low-density branches.

A key challenge is enforcing this stopping criterion under differential privacy. We cast the problem of identifying split nodes as a private heavy-hitters problem over a tree of height h. While related problems have been studied by Ghazi et al. [2023], Biswas et al. [2024], their techniques do not directly apply. A naive approach that adds independent noise to each node count suffers from poor utility for deep trees, due to ℓ_2 sensitivity (which determines the magnitude of noise needed for

[3] Unfortunately, the existing literature on differentially private random forests also includes several incorrect or flawed analyses. We found privacy issues in more than 10 papers, which we summarize in Appendix E. Despite the fact that these methods do not satisfy differential privacy as claimed, our approach still substantially outperforms their reported performance.

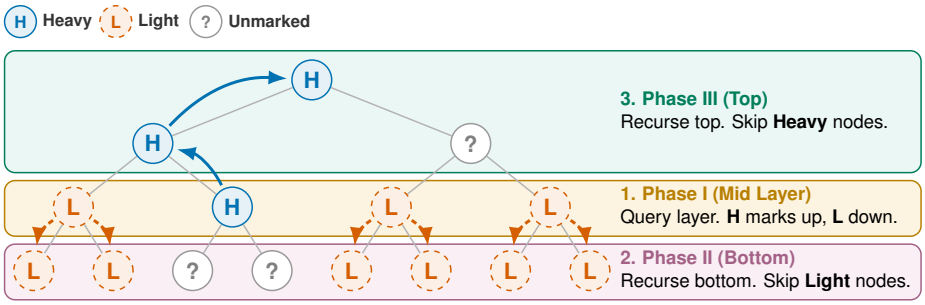

Figure 2: Example of a recursive iteration of our heavy hitter detector (Algorithm 1). **Phase I** (orange band) queries all nodes at the middle layer and marks them *Heavy* (**H**, blue) or *Light* (**L**, dashed orange), with implications for ancestors and descendants shown by arrows. **Phase II** (purple band) recursively explores subtrees containing unmarked nodes (**?**, gray), skipping *Light* nodes. **Phase III** (green band) recursively marks the upper half of the tree, skipping *Heavy* nodes.

privacy) scaling as $\sqrt{h}$. To overcome this, we exploit the monotone structure of rooted trees. Our key idea is to perform structured, highly overlapping noisy binary searches over the tree by querying nodes in a middle layer and propagating constraints upward and downward: heavy nodes imply heavy ancestors, while light nodes imply light descendants. As illustrated in Figure 2, this induces consistency constraints that reduce the problem to smaller independent subproblems of size at most $h/2$, reducing sensitivity to $\sqrt{1+\log_2 h}$. We further integrate techniques from sparse histogram estimation [Wilkins et al., 2024], improving both utility and computational efficiency. Finally, we develop a unified privacy analysis that bounds the privacy loss of the entire pruning procedure jointly over the forest construction, rather than relying on standard composition across trees, yielding a tighter privacy accounting.

Empirically, `Lumberjack` enables the training of deep, high-utility random forests under strict privacy constraints, achieving new state-of-the-art results and consistently outperforming existing baselines across multiple datasets and privacy regimes.

2 Preliminaries

Problem setup. We consider a dataset $D \in \mathcal{X}^n$ of n data points, where each point $x \in D$ has a set of features and a label $y \in \mathcal{Y}$. Features may be numerical or categorical; for numerical features we assume known bounds, and for categorical features a known domain. Our goal is to privately train a random forest to predict labels for unseen data. Random forests support both classification and regression; our contribution is a new differentially private tree-building algorithm applicable to both settings. We focus on classification in our presentation, as it is the primary setting in prior work.

Random forests. A Random Forest (RF) is an ensemble method that combines predictions from multiple decision trees. Each tree recursively partitions the feature space based on feature-value splits until a stopping criterion is reached (e.g., maximum depth). Each leaf corresponds to a region of the feature space and defines a score function $s : \mathcal{Y} \to [0, 1]$, typically derived from the training samples in that leaf. To predict, each tree outputs class scores, and the forest aggregates them (e.g., by summation) to select the class with highest total score. Common choices include majority vote, where each tree assigns all weight to a single class, or soft voting, where scores reflect class proportions within each leaf.

Differential privacy. Differential privacy [Dwork et al., 2006] (DP) is a framework for rigorous, quantifiable privacy guarantees. Two datasets $D \sim D'$ are neighbors if they differ by a single record (i.e., one addition or removal). Differential privacy requires that an algorithm's output distribution is approximately indistinguishable on any pair of neighboring datasets. Several variants have been proposed to suit different settings. In this work, we report privacy guarantees in (ε, δ)-DP and use zero-concentrated differential privacy (zCDP) for simpler privacy accounting.

Definition 1 ([Dwork and Roth, 2014] (ε, δ)-Differential Privacy)**.** *A randomized mechanism $\mathcal{M} : \mathcal{X}^* \to \mathcal{Z}$ satisfies (ϵ, δ)-DP, if for any $D \sim D'$ and every measurable set of outputs $Z \in \mathcal{Z}$ we have*

$$\Pr[\mathcal{M}(D) \in Z] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in Z] + \delta\,.$$

Remark 1. When $\delta = 0$, the guarantee is referred to as pure DP, or ε-DP.

Definition 2 ([Bun and Steinke, 2016] zero-Concentrated Differential Privacy)**.** *A randomized mechanism $\mathcal{M} : \mathcal{X}^* \to \mathcal{Z}$ satisfies ρ-zCDP, if for any $D \sim D'$ and all $\alpha \in (1, \infty)$, it holds that*

$$D_\alpha(\mathcal{M}(D) \parallel \mathcal{M}(D')) \leq \rho\alpha\,,$$

where $D_\alpha(\cdot \parallel \cdot)$ is the α-Rényi divergence between two distributions.

We present some standard useful lemmas for (ε, δ)-Differential Privacy and ρ-zCDP in Appendix A.

3 Related Work

Differentially private random forests. Existing DP-RF methods fall into two main paradigms.

The first paradigm [Patil and Singh, 2014, Fletcher and Islam, 2015a, Li and Li, 2017, Xin et al., 2019, Hou et al., 2019, Guan et al., 2020, Zhang et al., 2021, Consul and William, 2021, Vos et al., 2023, Suihkonen, 2023] allocates the privacy budget between private tree construction and leaf queries. Trees are built using impurity-based criteria similar to standard decision trees (e.g. Breiman [2001]), typically via the exponential mechanism [McSherry and Talwar, 2007]. We omit a detailed description due to the fact that our approach aligns more closely with the second paradigm, and because we identify privacy issues in all such works (see Appendix E).

The second paradigm, loosely inspired by extremely randomized trees ("Extra Trees") [Geurts et al., 2006], constructs trees via random splitting [Jagannathan et al., 2012, 2013, Fletcher and Islam, 2015b, 2017, Holohan et al., 2019]. In these methods, splits are chosen independently of the data. A key limitation of these techniques is that trees are fully expanded to a fixed depth without data-dependent stopping. While this avoids spending privacy budget on structure learning, these approaches poorly adapt to the data and often leads to degraded performance, as illustrated in Figure 1 and in our experiments of Section 6.

Our work departs from both paradigms by introducing a simple intermediate idea: we retain randomized splitting but use part of the privacy budget to determine when to stop splitting. Concretely, we build deep random trees and then privately prune them by identifying nodes that are sufficiently "heavy" in feature space; nodes whose parent is not heavy are removed. This allows data-dependent depth control without fully reverting to greedy tree construction.

Leaf prediction is handled either by adding noise to label counts (e.g. Fletcher and Islam [2015a]) or by private majority selection (e.g. [Fletcher and Islam, 2017]). We support both variants in our implementation (see Appendix J).

Remark 2. Most of the existing DP-RF methods rely on pure ε-DP, which is generally suboptimal for ensemble learning where privacy must be composed across many tree operations (e.g., repeated splits and leaf queries across many trees). Indeed, privacy costs compose linearly under ε-DP, leading to significantly higher noise compared to the approximate or concentrated DP variants we use in this work.

Differentially private heavy hitters in trees. A key technical component of our work is a novel algorithm for heavy-hitters detection in hierarchical data. This problem is related to several classical tasks in differential privacy, including quantile estimation [Kaplan et al., 2022, Huang et al., 2021, Bun et al., 2015, Cohen et al., 2023] and private release of sparse histograms [Korolova et al., 2009, Cormode et al., 2012, Balcer and Vadhan, 2019, Wilson et al., 2020, Aumüller et al., 2022, Lebeda and Retschmeier, 2025, Kerschbaum et al., 2025]. Another related problem is count estimation in trees, where noise scales with $O(\sqrt{h})$ with h the height of the tree. Despite extensive work on heavy hitters and hierarchical data release, there is limited work at their intersection. Ghazi et al. [2023] study a closely related problem (see Problems 4.2 and 4.4), achieving a reduced dependence of $O(\sqrt{\log h})$ at the cost of an additional $\sqrt{n}$ term, which is typically much larger than $\sqrt{h}$. Another relevant prior work is Biswas et al. [2024], who consider private release of Hierarchical Heavy Hitters (HHH), a strict generalization of our setting; indeed, our problem can be addressed by post-processing the output for the HHH problem. While their construction would be applicable to our setting, we identify a subtle error in their privacy analysis (see Appendix D). The authors recently uploaded a revised version of their paper to arXiv, correcting the mistake in the original version [Biswas et al., 2026]. Their corrected approach has worse asymptotic dependence than ours in terms of both h and δ, as well as larger constants. Consequently, our approach achieves substantially better performance in practice (see Table 4.1). We provide an expanded discussion of DP heavy hitters in Appendix B.

4 A Private Heavy Hitters Algorithm for Hierarchical Data

In this section, we present our novel private heavy hitters algorithm that will form the basis of our private random forest algorithm. We are given a tree structure $\mathcal{T}$ of height[4] h. Each data point contributes to the count of a leaf node in $\mathcal{T}$, and internal node counts are defined as the sum of their children's counts. Our goal is to privately identify all nodes whose count exceeds a threshold τ, while ensuring that the selected nodes form a valid subtree.

Baseline. A simple baseline traverses the tree in a top-down manner. At each node, we privately test whether its count exceeds the threshold τ, retaining nodes that pass and pruning subtrees whenever a node falls below the threshold. Since neighboring datasets affect counts along a single root-to-leaf path, the sensitivity is $\sqrt{h}$. We can thus apply the Gaussian mechanism to each count with noise scale $\sigma = \sqrt{h/(2\rho)}$ to satisfy ρ-zCDP. Our algorithm significantly improve upon this scaling in h.

4.1 Binary Search-Inspired Heavy Hitters Detection

High-level presentation of the approach. The pseudocode of our algorithm for finding heavy hitters in hierarchical data is given in Algorithm 1. The algorithm is parameterized by a private thresholding primitive CHECKTHRESHOLD, which privately tests whether a node count exceeds a threshold τ (it is "heavy"). The overall error of our method will be determined by the error of this primitive.

Conceptually, the algorithm performs 2^h highly overlapping noisy binary searches over the tree. To control privacy cost, nodes already identified as heavy by CHECKTHRESHOLD in one search are never queried again by another overlapping search. Figure 2 illustrates one iteration of the procedure. While noisy binary search is a classical tool for private quantile estimation in one dimension (e.g. the PrivQuant algorithm [Huang et al., 2021]), our main contribution is extending this idea to hierarchical tree structures and integrating it with sparse histogram techniques.

Our key observation is that each data point influences at most $1 + \lfloor \log_2 h \rfloor$ threshold queries, yielding an ℓ_2-sensitivity of $\sqrt{1 + \lfloor \log_2 h \rfloor}$. This is an exponential improvement over the $\sqrt{h}$ sensitivity of the standard top-down baseline. A naive implementation would still be impractical for deep trees due to the enormous number of empty nodes. To address this, we implement CHECKTHRESHOLD using techniques from the sparse histogram literature (Algorithm 2), allowing the algorithm to efficiently query only relevant (non-empty) nodes. This substantially improves both utility and computational efficiency. Finally, we derive tight (ε, δ)-DP guarantees by analyzing the privacy loss of the pruning procedure jointly, rather than through standard composition bounds.

Technical details and analysis. We begin by establishing some basic properties of Algorithm 1. Formal proofs are deferred to Appendix F; here we focus on the intuition behind the main results.

Lemma 3. *For any tree $\mathcal{T}$ with height h and any $x \in \mathcal{X}$, Algorithm 1 calls* CHECKTHRESHOLD *with at most $1 + \lfloor \log_2(h) \rfloor$ (equivalently $\lceil \log_2(h+1) \rceil$) nodes that contain x.*

Lemma 4. *The error of Algorithm 1 is bounded by the error of* CHECKTHRESHOLD*. Specifically, if all calls to* CHECKTHRESHOLD *where $\#count(u) \leq \tau - \alpha$ returns $\perp$ then all nodes in $\mathcal{T}$ with count $\leq \tau - \alpha$ are labeled Light. Similarly, if* CHECKTHRESHOLD *returns $\top$ for all calls where $\#count(u) \geq \tau + \beta$ then all nodes with count $\geq \tau + \beta$ are labeled Heavy.*

For Lemma 3, observe that each data point affects only a single query in Phase I and at most one recursive call of height at most $h/2$ (see Figure 2 for an illustration). This recursive structure directly yields the logarithmic sensitivity bound. For Lemma 4, we exploit the monotonicity of the tree. If a node is marked *Heavy* due to a heavy descendant, then the node itself must indeed be heavy since counts can only increase toward the root. Conversely, descendants of a *Light* node must also be light. These implications allow labels to propagate through the tree without additional queries.

By Lemma 3, Algorithm 1 satisfies ρ-zCDP if we instantiate CHECKTHRESHOLD with the Gaussian mechanism, analogous to the baseline but with ℓ_2-sensitivity reduced to $\sqrt{1 + \lfloor \log_2 h \rfloor}$ (see Lemma 14). However, a direct implementation remains impractical for deep trees, since Phase I alone queries $2^{\lfloor h/2 \rfloor}$ nodes. This leads not only to prohibitive computational and memory costs,

[4]We define the height as the number of nodes on a root-to-leaf path, including both endpoints. An alternative convention counts edges instead and thus differs by 1. We use depth to refer to this alternative convention. Thus, height($\mathcal{T}$) = depth($\mathcal{T}$) + 1.

Algorithm 1 MARKHEAVYHITTERS($\mathcal{T}$, h)

Require: A tree $\mathcal{T}$ with height h and a threshold $\tau \in \mathbb{R}$.
Require: A DP algorithm CHECKTHRESHOLD$_\tau : \mathbb{N} \to \{\top, \perp\}$.
{Phase 1: Query the middle layer}
Let L_{mid} be the set of nodes at level $\lfloor h/2 \rfloor$ of $\mathcal{T}$.
for all node $u \in L_{\text{mid}}$ where u has not been marked **do**
  if CHECKTHRESHOLD$_\tau(\#count(u)) = \top$ **then**
    Mark u and all ancestors of u as *Heavy*.
  else
    Mark u and all descendants of u as *Light*.
  end if
end for
{Phase 2: Recursively apply the algorithm to children that were not marked *Light* in Phase 1}
if $h \geq 3$ **then**
  Let $L_{\text{mid}+1}$ be the set of nodes at level $\lfloor h/2 \rfloor + 1$ of $\mathcal{T}$.
  for all node $v \in L_{\text{mid}+1}$ where v is NOT marked *Light* **do**
    Let $\mathcal{T}_v$ be the subtree of $\mathcal{T}$ rooted at v.
    MARKHEAVYHITTERS($\mathcal{T}_v$, $h - \lfloor h/2 \rfloor - 1$).
  end for
end if
{Phase 3: Recursive apply the algorithm in the top part of the tree}
if $h \geq 2$ **then**
  Let $\hat{\mathcal{T}}$ be $\mathcal{T}$ restricted to layers $\{0, \ldots, \text{mid} - 1\}$.
  MARKHEAVYHITTERS($\hat{\mathcal{T}}$, $\lfloor h/2 \rfloor$).
end if

but also to error scaling as $O(\sigma\sqrt{\ln(2^{\lfloor h/2 \rfloor})}) = O_{\varepsilon,\delta}(\sqrt{h \log h})$. Since most of these nodes are empty whenever $h/2 > \log_2 n$, we instead adapt techniques from sparse histogram estimation. Specifically, we introduce an adaptive variant of the method from Wilkins et al. [2024] to implement CHECKTHRESHOLD (see Algorithm 2). Threshold queries with sufficiently small counts are deterministically mapped to $\perp$ (capturing all empty nodes under suitable parameters), while Gaussian noise is added only for the remaining queries before comparing against the threshold.

Algorithm 2 GaussianSparseThreshold

Require: Parameters σ, τ, and Δ.
Require: Dataset D, adaptive counting queries $q_1, q_2, q_3, \ldots$
for all $i \in \{1, 2, 3, \ldots\}$ **do**
  Sample $Z_i \sim \mathcal{N}\left(0, \sigma^2\right)$.
  if $q_i(D) > \tau - \Delta - 1$ and $q_i(D) + Z_i > \tau$ **then**
    Release $\top$.
  else
    Release $\perp$.
  end if
end for

We present parameter settings for Algorithm 2 that ensure (ε, δ)-DP in Appendix G, and summarize them as part of our main theoretical result below. We next analyze the accuracy of the mechanism, measuring error in terms of worst-case misclassification. The following lemma follows from standard Gaussian tail bounds.

Lemma 5. *Let $\alpha \in \mathbb{R}$ be the smallest value such that all nodes with count at least $\tau + \alpha$ are marked Heavy by Algorithm 1, and all nodes with count at most $\tau - \alpha$ are marked Light. If* CHECKTHRESHOLD *is implemented via Algorithm 2 with parameters $\sigma > 0$ and $\tau \geq 1 + \Delta$, then with probability at least $1 - \beta$, we have*

$$\alpha \leq \sigma \cdot \sqrt{2 \ln(2n\lceil \log_2(h+1) \rceil / \beta)} \ .$$

We are now ready to present our main theoretical result. In contrast to prior work, which applies a composition theorem across trees to obtain privacy guarantees for the full forest, we directly analyze the privacy of the entire forest construction. This avoids the loss introduced by repeated composition under (ε, δ)-DP and yields a tighter overall bound.

Theorem 6. *Let $\mathcal{A}$ denote the algorithm that takes as input k trees with maximum height h, and runs Algorithm 1 where* CHECKTHRESHOLD *is instantiated as Algorithm 2 with parameters σ, τ, and Δ. Then $\mathcal{A}$ satisfies (ε, δ)-DP where $m = k(1 + \lfloor \log_2(h) \rfloor)$, $\gamma(j) = (m - j) \log \Phi\left(\frac{\Delta}{\sigma}\right)$, and*

$$\delta \geq \max \Bigg[1 - \Phi\left(\frac{\Delta}{\sigma}\right)^m,$$
$$\max_{j \in [m]} 1 - \Phi\left(\frac{\Delta}{\sigma}\right)^{m-j} + \Phi\left(\frac{\Delta}{\sigma}\right)^{m-j} \left[\Phi\left(\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon - \gamma(j))\sigma}{\sqrt{j}}\right) - e^{\varepsilon - \gamma(j)} \Phi\left(-\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon - \gamma(j))\,\sigma}{\sqrt{j}}\right) \right],$$
$$\max_{j \in [m]} \Phi\left(\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon + \gamma(j))\,\sigma}{\sqrt{j}}\right) - e^{\varepsilon + \gamma(j)} \Phi\left(-\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon + \gamma(j))\sigma}{\sqrt{j}}\right) \Bigg].$$

If $\tau \geq 1 + \Delta$, then with probability at least $1 - \beta$, the algorithm correctly classifies all nodes with count $\leq \tau - \alpha$ as Light and all nodes with count $\geq \tau + \alpha$ as Heavy where

$$\alpha \leq \sigma \cdot \sqrt{2 \ln(2kn\lceil \log_2(h+1) \rceil / \beta)}\ .$$

Furthermore, a variant of the algorithm that returns all nodes marked Heavy can be implemented in time and space $O(knh)$.

Proof. By Lemma 3 adding or removing a data point affects at most $1 + \lfloor \log(h) \rfloor$ queries for each tree. The privacy guarantees thus follow from Lemma 15 (see Appendix G). The error bound holds for each tree with probability at least $1 - \beta/k$ by Lemma 5, and thus with probability at least $1 - \beta$ it holds for the entire forest simultaneously. For any $\tau \geq 1 + \Delta$, Algorithm 2 returns $\perp$ for all empty nodes. We may therefore avoid inspecting subtrees of empty nodes as we know these nodes would all be marked as *Light*. This does not affect the output distribution and thus preserves the DP guarantee. It is easy to see that there are at most $O(nh)$ non-empty nodes in each tree. We discuss implementation details to achieve space and running time linear in the number of non-empty nodes in Appendix J. □

We use the parameters presented in Theorem 6 in our implementation, although the resulting privacy expressions are not very intuitive. For intuition, we provide below a simpler but looser closed-form bound in the regime $\varepsilon < 1$, which makes explicit the dependence on h, k, σ, and Δ.

Lemma 7. *Define $\mathcal{A}$ as in Theorem 6. If $\varepsilon < 1$, the parameters*

$$\sigma = \frac{\sqrt{2m \ln(2.5/\delta)}}{\varepsilon} \qquad \Delta = \sigma \cdot \sqrt{2 \ln(2m/\delta)}$$

satisfy the condition in Theorem 6 for (ε, δ)-DP where $m = k(1 + \lfloor \log_2 h \rfloor)$.

Remark 3. We give high probability bounds for the maximum error in Lemma 5 and Theorem 6. However, notice that the error for nodes with small counts is deterministically bounded by Δ. We can achieve a deterministic error bound using a variant of Algorithm 2 as discussed in Appendix H.

Comparison with prior work. Our technique has better asymptotic scaling than prior techniques [Ghazi et al., 2023, Biswas et al., 2026] and, importantly, also much smaller constants. As a result, we outperform both approaches by several orders of magnitude across most practical parameter regimes. To illustrate this gap, we compare the minimum value of τ required by each technique to ignore empty nodes when finding heavy hitters ($\tau = 1 + \Delta$). We consider a single tree, fix $\delta = 10^{-6}$ and $h = 100$, and vary ε. Both [Ghazi et al., 2023] and [Biswas et al., 2026] require a failure probability as input; we conservatively set it to $1/4$ for both methods (smaller values would further increase their required τ). The error of [Ghazi et al., 2023] also scales with the dataset size. We set $n = 100\,000$, which is comparable to the dataset sizes used in our experiments. The results in Section 4.1 show the clear superiority of our approach.

ε	0.1	0.5	1	4
Lumberjack (Theorem 6)	494	110	58	17
Ghazi et al. [2023]	25 332	11 328	8 011	4 005
Biswas et al. [2026]	10 402 352	1 654 660	735 637	138 062

Table 1: Smallest valid τ for a single tree ($\delta = 10^{-6}$).

4.2 Lower Bound for Deep Trees

By Theorem 6, we should set the threshold τ to at least $1+\Delta$. This ensures an efficient implementation for deep trees and improves utility by allowing us to deterministically label empty nodes. As a result, the threshold may be larger than what would be chosen in the non-private setting (e.g., in random forests), but we show below that this is unavoidable for any private method that supports deep trees.

We now outline structural properties of heavy hitters algorithms that lead to our lower bound. First, observe that Algorithm 1 outputs at most n nodes per layer when $\tau \geq 1 + \Delta$. While this is desirable, we allow for a broader class of algorithms that may output up to n^2 nodes per layer, since slightly larger memory usage can still be practical. Any method exceeding this regime quickly becomes infeasible for moderate or large datasets, though the lower bound still applies for e.g. n^3 nodes with slightly different constants. We additionally require that all sufficiently heavy nodes are recovered with high probability. Together, these conditions yield the following lower bound.

Lemma 8 (Informal)**.** *Let $\mathcal{M}$ denote an (ε, δ)-DP mechanism (for any $\varepsilon > \delta^2$) that takes as input a tree $\mathcal{T}$ of height $h > 4\log(n)$ and a dataset of size n, and outputs a set of at most n^2 nodes per layer of $\mathcal{T}$. Suppose that, with high probability, $\mathcal{M}$ outputs all nodes in $\mathcal{T}$ with count above some value $\hat{\tau} \in \mathbb{R}$. Then $\hat{\tau} \geq \Omega\left(\frac{\min(h, \log(1/\delta))}{\varepsilon}\right)$.*

The lemma follows from a lower bound for sparse histograms [Balcer and Vadhan, 2019, Theorem 7.2], which (informally) states that any (ε, δ)-DP mechanism that outputs few nonzero bins must incur error $\Omega\left(\frac{\min(\log d, \log(1/\delta))}{\varepsilon}\right)$ for d-dimensional histograms. We can encode such a histogram over any $d \leq 2^{h-1}$ in the leaf level of the tree. Technical details are deferred to Appendix I.

Note that our algorithm is at most a factor $O(\sqrt{\log h})$ from optimal for $\varepsilon < 1$ (see Corollaries 17 and 18). We emphasize that tight lower bounds are not the main focus of this work, and leave closing this gap as an open problem.

5 `Lumberjack`: Our Differentially Private Random Forest Method

In this section, we describe how we use the heavy hitters algorithm from the previous section to instantiate our random forest method, `Lumberjack`. We split the algorithm into two components: one for privately constructing the tree structures and one for privatizing the leaf predictions. We ensure that constructing all trees together satisfies $(\varepsilon_1, \delta_1)$-DP and that privatizing all leaf nodes across all trees satisfies $(\varepsilon_2, \delta_2)$-DP. By basic composition, `Lumberjack` therefore satisfies $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-DP.

Tree construction. Given a maximum depth d, we recursively partition the feature space until this depth is reached. We then pass the resulting tree, restricted to nodes with depth $\{0, \ldots, d-1\}$ (corresponding to height $h = d$) to our heavy hitters algorithm. The heavy hitters returned by the algorithm are retained as split nodes in the final tree. Any node whose parent is labeled *Light* is pruned, and the remaining *Light* nodes become the leaf nodes.

To split the feature space, we select a feature uniformly at random. For numerical features, we assume access to a range $[l, u]$ approximating the support of that feature (this is a standard assumption in DP). We then sample a threshold uniformly at random, $t \in [l, u]$, and split the node accordingly. The ranges for the left and right child nodes are updated to $[l, t]$ and $[t, u]$, respectively. Categorical features are encoded using one-hot encoding, following common practice in non-private implementations (e.g., `sklearn` [Pedregosa et al., 2011]). When splitting on a categorical feature, we select a category that has not already been used in an ancestor node. Importantly, the privacy guarantees of `Lumberjack` are independent of the specific strategy used to generate random splits. Exploring alternative splitting schemes together with (privacy-preserving) data preprocessing is an interesting direction for improving utility in future work.

Remark 4. Many non-private random forest algorithms train each tree on a subsample of the data using bootstrap sampling or sampling without replacement [Pedregosa et al., 2011, Athey et al., 2019], which improves diversity and reduces overfitting. In contrast, `Lumberjack` uses the full dataset for each tree, with randomness naturally coming from the splitting procedure, as in non-private Extra Trees [Geurts et al., 2006].

Leaf predictions. The last step is to privatize predictions in leaf nodes. For simplicity, we focus on classification in the main text and defer the extension to regression, along with some illustrative experiments, to Appendix K.We consider two mechanisms for privatizing leaf predictions. First, we support the exponential mechanism to select a single class label for each leaf. The prediction for the entire forest then consists of a majority vote across trees. Second, we support adding noise to class counts to obtain a private approximation of the class proportions, and average these proportions across trees. We did not observe significant performance differences between the two approaches in initial exploratory experiments; majority voting is used in our experiments. We perform privacy accounting in zCDP. Specifically, we compute the largest ρ such that any ρ-zCDP algorithm satisfies $(\varepsilon_2, \delta_2)$-DP (see Lemma 9). We then apply a (ρ/k)-zCDP mechanism independently to each leaf, which yields an overall ρ-zCDP guarantee for the forest because each data point influences only a single leaf per tree. Additional implementation details are provided in Appendix J.

Choosing hyperparameters. We recommend a simple, principled rule for setting τ, Δ, and σ that requires no hyperparameter tuning and consistently delivers strong performance across our experiments. Specifically, given a privacy budget (ε, δ), a number of trees k, and a maximum height h, we choose the valid pair (Δ, σ) satisfying the inequality in Theorem 6 that minimizes Δ, and set $\tau = \Delta + 1$. This threshold is typically higher than common used values in the non-private setting. Increasing τ further may be beneficial for very large datasets or when using weaker privacy guarantees. Since pruning adaptively determines the effective depth of each tree, the maximum depth h should be set sufficiently large so that it rarely becomes a limiting factor. In our experiments, setting $h = 100$ works well across all datasets. Unlike the non-private setting, the number of trees k directly affects the quality of individual trees, since the privacy budget must be divided among all trees. Larger forests therefore consist of coarser and noisier trees. Nevertheless, we empirically find that accuracy remains stable across a broad range of forest sizes. Finally, one must choose how to split the privacy budget between tree construction and leaf queries. We observe stable performance across a broad range of allocations. For classification, we recommend allocating most of the privacy budget to tree construction. For regression, leaf queries are arguably more challenging to estimate, so we allocate a larger share of the budget to them (see Appendix K).

6 Experiments

Baselines. We compare `Lumberjack` against several differentially private random forest methods, including three randomized tree-based approaches (IBM's Differential Privacy Library `DiffPrivLib` [Holohan et al., 2019], `SNR` [Fletcher and Islam, 2015b] and `Smooth Sensitivity` [Fletcher and Islam, 2017]), and one greedy tree method (`Suihkonen` [Suihkonen, 2023]). As non-private baselines, we use `sklearn`'s Extra Trees and Decision Tree classifiers [Pedregosa et al., 2011]. We evaluate all methods across privacy budgets $\varepsilon \in \{0.5, 1, 2, 4, 8\}$ with $\delta = 10^{-6}$ for `Lumberjack`. We note that some baselines have privacy issues, leading to weaker-than-claimed guarantees; this appears to be a recurring issue in prior work on DP random forests (see Appendix E).

Datasets. We evaluate performance on the `Adult` dataset [Becker and Kohavi, 1996] and four binary classification tasks from `Folktables` (California 2018) [Ding et al., 2021]. We exclude ACSMobility due to uniformly poor performance across all methods, including non-private baselines. Dataset statistics are summarized in Table 2. Each experiment is repeated 5 times and results are averaged.

Hyperparameters. We use an 80-10-10 train/validation/test split. Hyperparameters for `DiffPrivLib` and non-private baselines are tuned on the validation set, without accounting for additional privacy cost. The performance of `DiffPrivLib` is sensitive to these choices. For `SNR`, `Smooth Sensitivity`, and `Suihkonen`, we use the settings recommended in the original papers. For `Lumberjack`, we fix a maximum tree depth of 100 across all experiments. We train 30 trees on `Adult` and 50 trees on all `Folktables` tasks. We allocate 75% of the privacy budget to tree construction ($\varepsilon_1 = 0.75\varepsilon$). We set $\tau = 1 + \Delta$ in all experiments.

Table 2: Summary statistics for the Adult and Folktables (CA) datasets.

Dataset	Features	Categorical	Classes	Majority	Datapoints
`Adult`	14	8	2	76.1 %	48,842
`ACSIncome (CA)`	10	4	2	58.9 %	195,665
`ACSEmployment (CA)`	16	10	2	54.4 %	378,817
`ACSPublicCoverage (CA)`	19	10	2	63.1 %	138,554
`ACSTravelTime (CA)`	16	7	2	51.5 %	172,508

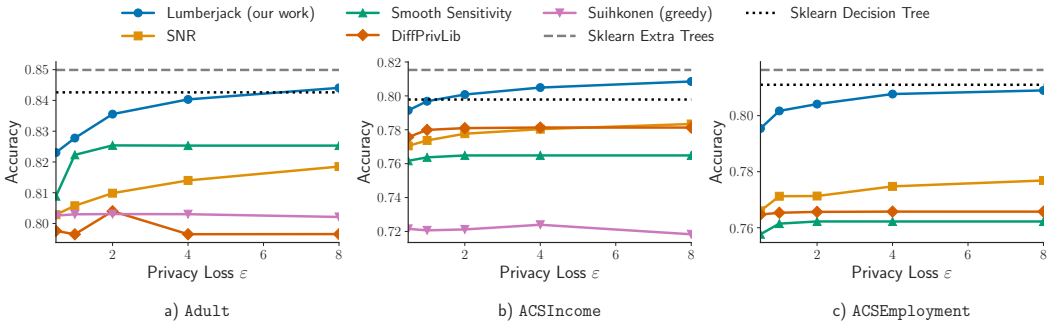

Figure 3: Test accuracy for the Adult and Folktables ACSIncome and ACSEmployment datasets. We exclude the Suihkonen baseline from ACSEmployment because its runtime was too high.

Results. As shown in Figure 3 (and Figure 4 in the appendix), `Lumberjack` consistently and significantly outperforms prior work across all evaluated privacy budgets. `DiffPrivLib` and `Smooth Sensitivity` achieve reasonable performance at small ε but do not improve as privacy relaxes. This is partly due to their reliance on data partitioning for leaf predictions, which allows them to perform accurate leaf predictions at small privacy budgets but shifts the burden to random tree construction. We also note that `Smooth Sensitivity` handles categorical features differently than `DiffPrivLib`: it splits on *all* values of a categorical feature instead of 1, which provides an advantage on `Adult` but does not generalize well to the other datasets. `SNR` outperforms the other baselines for some datasets but remains substantially below `Lumberjack`. `Suihkonen` matches previously reported results on `Adult` [Suihkonen, 2023] but performs poorly on the `Folktables` tasks.

While a gap remains to non-private `Extra Trees`, `Lumberjack` often outperforms a non-private greedy decision tree even at moderate privacy levels. Overall, `Lumberjack` establishes a new state-of-the-art among DP random forest methods. Additional results are provided in Appendix C. Source code is available at `https://github.com/daviderb/Lumberjack`.

7 Conclusion

Despite the popularity of random forests in non-private machine learning, their differentially private counterparts remain relatively underdeveloped and, in many cases, problematic in both correctness and utility. Our work takes a step toward addressing these issues. We introduce `Lumberjack`, a new DP random forest method based on a combination of randomized tree construction and data-dependent pruning based on a novel heavy hitters algorithm which may be of independent interest. Empirically, `Lumberjack` sets a new state-of-the-art in DP random forest methods.

We believe that designing high-utility differentially private greedy random forests remains an open and promising direction. Achieving this goal will likely require more careful integration of advanced tools from the DP literature, such as refined composition techniques, tighter privacy accounting, and problem-specific mechanisms that better exploit the structure of greedy splitting procedures.

Overall, we hope our work helps clarify the design space and current status of DP random forests, and encourages further research on methods that more closely match the performance of their non-private counterparts while maintaining rigorous privacy guarantees.

Acknowledgements

The work of David Erb was carried out while visiting the PreMeDICaL Inria team. The work of Christian Janos Lebeda, Tudor Cebere, and Aurélien Bellet is supported by grant ANR-20-CE23-0015 (Project PRIDE) and the ANR 22-PECY-0002 IPOP (Interdisciplinary Project on Privacy) project of the Cybersecurity PEPR. The work of Tudor Cebere is also supported by a Google PhD Fellowship in Privacy, Safety, and Security. The work of Christian Janos Lebeda is also supported by the Carlsberg Foundation, grant CF25-1559. This work was performed using HPC resources from GENCI–IDRIS (Grant 2023-AD011014018R2).

References

Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, October 2016. doi: 10.1145/2976749.2978318. URL `http://arxiv.org/abs/1607.00133`. arXiv:1607.00133 [cs, stat].

John M. Abowd, Robert Ashmead, Ryan Cumings-Menon, Simson L. Garfinkel, Micah Heineck, Christine Heiss, Robert Johns, Daniel Kifer, Philip Leclerc, Ashwin Machanavajjhala, Brett Moran, William Sexton, Matthew Spence, and Pavel Zhuravlev. The 2020 census disclosure avoidance system topdown algorithm. *Harvard Data Science Review*, 2, 2022.

Susan Athey, Julie Tibshirani, and Stefan Wager. Generalized random forests. 2019.

Martin Aumüller, Christian Janos Lebeda, and Rasmus Pagh. Representing sparse vectors with differential privacy, low error, optimal space, and fast access. *Journal of Privacy and Confidentiality*, 12(2), Nov. 2022. doi: 10.29012/jpc.809. URL `https://journalprivacyconfidentiality.org/index.php/jpc/article/view/809`.

Victor Balcer and Salil P. Vadhan. Differential privacy on finite computers. *J. Priv. Confidentiality*, 9 (2), 2019.

Borja Balle and Yu-Xiang Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 403–412. PMLR, 2018.

Barry Becker and Ronny Kohavi. Adult, 1996. URL `https://doi.org/10.24432/C5XW20`.

Giulia Bernardini, Philip Bille, Inge Li Gørtz, and Teresa Anna Steiner. Differentially private substring and document counting. *Proc. ACM Manag. Data*, 3(2):95:1–95:27, 2025.

Ari Biswas, Graham Cormode, Yaron Kanza, Divesh Srivastava, and Zhengyi Zhou. Differentially private hierarchical heavy hitters. *Proc. ACM Manag. Data*, 2(5):208:1–208:25, 2024.

Ari Biswas, Graham Cormode, Yaron Kanza, Divesh Srivastava, and Zhengyi Zhou. Differentially private hierarchical heavy hitters. *CoRR*, abs/2606.13563, 2026. doi: 10.48550/ARXIV.2606.13563. URL `https://doi.org/10.48550/arXiv.2606.13563`.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*, pages 635–658. Springer, 2016.

Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. Differentially private release and learning of threshold functions. In *FOCS*, pages 634–649. IEEE Computer Society, 2015.

Sílvia Casacuberta, Michael Shoemate, Salil P. Vadhan, and Connor Wagaman. Widespread underestimation of sensitivity in differentially private libraries and how to fix it. In *CCS*, pages 471–484. ACM, 2022.

Tudor Cebere, David Erb, Damien Desfontaines, Aurélien Bellet, and Jack Fitzsimons. Privacy in theory, bugs in practice: Grey-box auditing of differential privacy libraries. *CoRR*, abs/2602.17454, 2026.

Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, and Uri Stemmer. Optimal differentially private learning of thresholds and quasi-concave optimization. In *STOC*, pages 472–482. ACM, 2023.

Shorya Consul and Sinead A William. Differentially private random forests for regression and classification. *Association for the Advancement of Artificial Intelligence*, 2021.

Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Thanh T. L. Tran. Differentially private summaries for sparse data. In *ICDT*, pages 299–311. ACM, 2012.

Matt Dawson, Badih Ghazi, Pritish Kamath, Kapil Kumar, Ravi Kumar, Bo Luan, Pasin Manurangsi, Nishanth Mundru, Harikesh Nair, Adam Sealfon, and Shengyu Zhu. Optimizing hierarchical queries for the attribution reporting api. In *AdKDD@KDD*, 2023. URL `https://ceur-ws.org/Vol-3556/adkdd23-dawson-optimizing-ceur-paper.pdf`.

Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Jinshuo Dong, Aaron Roth, and Weijie J Su. Gaussian differential privacy. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 84(1):3–37, 2022.

Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 3rd Conference on Theory of Cryptography*, TCC '06, pages 265–284, Berlin, Heidelberg, 2006. Springer.

Alexander Edmonds, Aleksandar Nikolov, and Jonathan R. Ullman. The power of factorization mechanisms in local and central differential privacy. In *STOC*, pages 425–438. ACM, 2020.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1):3133–3181, 2014.

Julien Ferry, Ricardo Fukasawa, Timothée Pascal, and Thibaut Vidal. Trained random forests completely reveal your dataset. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 13545–13569. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/ferry24a.html`.

Sam Fletcher and Md Zahidul Islam. A differentially private decision forest. In *AusDM*, volume 168 of *CRPIT*, pages 99–108. Australian Computer Society, 2015a.

Sam Fletcher and Md Zahidul Islam. A differentially private random decision forest using reliable signal-to-noise ratios. In *Australasian Conference on Artificial Intelligence*, volume 9457 of *Lecture Notes in Computer Science*, pages 192–203. Springer, 2015b.

Sam Fletcher and Md Zahidul Islam. Differentially private random decision forests using smooth sensitivity. *Expert systems with applications*, 78:16–31, 2017.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Badih Ghazi, Pritish Kamath, Ravi Kumar, Pasin Manurangsi, and Kewen Wu. On differentially private counting on trees. In *ICALP*, LIPIcs, pages 66:1–66:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

Badih Ghazi, Pritish Kamath, Ravi Kumar, Pasin Manurangsi, and Adam Sealfon. Denoising the us census: Succinct block hierarchical regression. *arXiv preprint arXiv:2603.10099*, 2026.

Michaela Gotz, Ashwin Machanavajjhala, Guozhang Wang, Xiaokui Xiao, and Johannes Gehrke. Publishing search logs—a comparative study of privacy guarantees. *IEEE Transactions on Knowledge and Data Engineering*, 24(3):520–532, 2012. doi: 10.1109/TKDE.2011.26.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Zhitao Guan, Xianwen Sun, Lingyun Shi, Longfei Wu, and Xiaojiang Du. A differentially private greedy decision forest classification algorithm with high utility. *Comput. Secur.*, 96:101930, 2020.

Peaker Guo, Rayne Holland, and Hao Wu. Fast and optimal differentially private frequent-substring mining. *CoRR*, abs/2603.09166, 2026.

Guanlin He, Yingtai Xiao, Jiamu Bai, Xin Gu, Zeyu Ding, Wenpeng Yin, and Daniel Kifer. Accurate and scalable matrix mechanisms via divide and conquer. *arXiv preprint arXiv:2604.00868*, 2026.

Naoise Holohan, Stefano Braghin, Pól Mac Aonghusa, and Killian Levacher. Diffprivlib: The ibm differential privacy library, 2019. URL `https://arxiv.org/abs/1907.02444`.

James Honaker. Efficient use of differentially private binary trees. *Theory and Practice of Differential Privacy (TPDP 2015), London, UK*, 2:26–27, 2015.

Jun Hou, Qianmu Li, Shunmei Meng, Zhen Ni, Yini Chen, and Yaozong Liu. DPRF: A differential privacy protection random forest. *IEEE Access*, 7:130707–130720, 2019.

Ziyue Huang, Yuting Liang, and Ke Yi. Instance-optimal mean estimation under differential privacy. In *NeurIPS*, pages 25993–26004, 2021.

Geetha Jagannathan, Krishnan Pillaipakkamnatt, and Rebecca N. Wright. A practical differentially private random decision tree classifier. *Trans. Data Priv.*, 5(1):273–295, 2012.

Geetha Jagannathan, Claire Monteleoni, and Krishnan Pillaipakkamnatt. A semi-supervised learning approach to differential privacy. In *ICDM Workshops*, pages 841–848. IEEE Computer Society, 2013.

Kaggle. 2022 Kaggle Machine Learning & Data Science Survey, 2022. URL `https://www.kaggle.com/c/kaggle-survey-2022`.

Haim Kaplan, Shachar Schnapp, and Uri Stemmer. Differentially private approximate quantiles. In *International Conference on Machine Learning*, pages 10751–10761. PMLR, 2022.

Florian Kerschbaum, Steven Lee, and Hao Wu. Optimal pure differentially private sparse histograms in near-linear deterministic time. *CoRR*, abs/2507.17017, 2025.

Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. Releasing search queries and clicks privately. In *WWW*, pages 171–180. ACM, 2009. doi: 10.1145/1526709.1526733.

Alex Kulesza, Ananda Theertha Suresh, and Yuyan Wang. Mean estimation in the add-remove model of differential privacy. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 25642–25661. PMLR / OpenReview.net, 2024. URL `https://proceedings.mlr.press/v235/kulesza24a.html`.

Christian Janos Lebeda and Lukas Retschmeier. The correlated gaussian sparse histogram mechanism. In *FORC*, LIPIcs, pages 23:1–23:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

Christian Janos Lebeda, Aleksandar Nikolov, and Haohua Tang. Weighted fourier factorizations: Optimal gaussian noise for differentially private marginal and product queries. *CoRR*, abs/2512.21499, 2025.

Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *VLDB J.*, 24(6):757–781, 2015.

Zekun Li and Shuyu Li. Random forest algorithm under differential privacy. In *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pages 1901–1905. IEEE, 2017.

Jing Liu, Xianxian Li, Quanmin Wei, Songfeng Liu, Zhaohui Liu, and Jinyan Wang. A two-phase random forest with differential privacy. *Appl. Intell.*, 53(10):13037–13051, 2023.

Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. *arXiv preprint arXiv:1603.01699*, 2016.

Ryan McKenna and Daniel Sheldon. Permute-and-flip: A new mechanism for differentially private selection. In *NeurIPS*, 2020.

Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103. IEEE Computer Society, 2007.

R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33 (3):291–297, 1997.

Nicolas Papernot and Thomas Steinke. Hyperparameter tuning with renyi differential privacy. In *ICLR*. OpenReview.net, 2022.

Abhijit Patil and Sanjay Singh. Differential private random forest. In *ICACCI*, pages 2623–2630. IEEE, 2014.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H. Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Guha Thakurta. How to dp-fy ML: A practical guide to machine learning with differential privacy. *J. Artif. Intell. Res.*, 77:1113–1201, 2023. doi: 10.1613/JAIR.1.14649. URL https://doi.org/10.1613/jair.1.14649.

Ryan Rogers and Thomas Steinke. A better privacy analysis of the exponential mechanism. DifferentialPrivacy.org, 07 2021. https://differentialprivacy.org/exponential-mechanism-bounded-range/.

Sini Suihkonen. Differential privacy applied to random forest classification. Master's thesis, University of Helsinki, November 2023. URL https://helda.helsinki.fi/handle/10138/. Master's Programme in Data Science.

Florian Tramèr and Dan Boneh. Differentially Private Learning Needs Better Features (or Much More Data), February 2021. URL http://arxiv.org/abs/2011.11660. arXiv:2011.11660 [cs, stat].

Shahadat Uddin and Haohui Lu. Confirming the statistically significant superiority of tree-based machine learning algorithms over their counterparts for tabular data. *PLOS ONE*, 19(4):1–12, 04 2024. doi: 10.1371/journal.pone.0301541. URL https://doi.org/10.1371/journal.pone.0301541.

Daniël Vos, Jelle Vos, Tianyu Li, Zekeriya Erkin, and Sicco Verwer. Differentially-private decision trees and provable robustness to data poisoning. *arXiv preprint arXiv:2305.15394*, 2023.

Arjun Wilkins, Daniel Kifer, Danfeng Zhang, and Brian Karrer. Exact privacy analysis of the gaussian sparse histogram mechanism. *J. Priv. Confidentiality*, 14(1), 2024.

Royce J. Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private SQL with bounded user contribution. *Proc. Priv. Enhancing Technol.*, 2020(2):230–250, 2020. doi: 10.2478/POPETS-2020-0025. URL https://doi.org/10.2478/popets-2020-0025.

Yingtai Xiao, Guanlin He, Danfeng Zhang, and Daniel Kifer. An optimal and scalable matrix mechanism for noisy marginals under convex loss functions. In *NeurIPS*, 2023.

Bangzhou Xin, Wei Yang, Shaowei Wang, and Liusheng Huang. Differentially private greedy decision forest. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2672–2676. IEEE, 2019.

Yaling Zhang, Pengfei Feng, and Yao Ning. Random forest algorithm based on differential privacy protection. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1259–1264. IEEE, 2021.

A Additional Preliminaries

Here we present standard technical results from differential privacy.

Any ρ-zCDP guarantee can be converted to (ε, δ)-DP guarantees as follows.

Lemma 9 ([Bun and Steinke, 2016])**.** *Let $\mathcal{M} : \mathcal{X}^* \to \mathcal{Z}$ be a randomized mechanism that satisfies ρ-zCDP. Then, $\mathcal{M}$ satisfies (ε, δ)-DP. for all $\varepsilon \geq \rho$ and*

$$\delta = \frac{2e^{-(\varepsilon-\rho)^2/4\rho}}{1 + \frac{\varepsilon-\rho}{2\rho} + \sqrt{(1+\frac{\varepsilon-\rho}{2\rho})^2 + \frac{4}{\pi\rho}}}.$$

The consecutive execution of differentially private mechanisms yields another differentially private mechanism. We use the following composition theorems.

Lemma 10 ([Dwork and Roth, 2014] Basic Composition)**.** *Let $\mathcal{M}_1 : \mathcal{X}^* \to \mathcal{Z}_1, \mathcal{M}_2 : \mathcal{X}^* \times \mathcal{Z}_1 \to \mathcal{Z}_2$ be randomized mechanisms that satisfy $(\varepsilon_1, \delta_1)$-DP and $(\varepsilon_2, \delta_2)$-DP respectively, then the composed mechanism $\mathcal{M}(D) = \mathcal{M}_2(D, \mathcal{M}_1(D))$ satisfies $(\varepsilon_1 + \varepsilon_2, \delta_1 + \delta_2)$-DP.*

Lemma 11 ([Bun and Steinke, 2016] zCDP Composition)**.** *Let $\mathcal{M}_1 : \mathcal{X}^* \to \mathcal{Z}_1, \mathcal{M}_2 : \mathcal{X}^* \times \mathcal{Z}_1 \to \mathcal{Z}_2$ be randomized mechanisms that satisfy ρ_1-zCDP and ρ_2-zCDP respectively, then the composed mechanism $\mathcal{M}(D) = \mathcal{M}_1(D, \mathcal{M}_2(D))$ satisfies $\rho_1 + \rho_2$-zCDP.*

A key property of differential privacy is that the usage of the output of a private mechanism in any computation cannot make the output less private.

Definition 12 ([Dwork and Roth, 2014] Post-Processing)**.** *A randomized mechanism $\mathcal{M} : \mathcal{X}^* \to \mathcal{Z}$ with privacy guarantee g is immune to post-processing if for any randomized mapping $f : \mathcal{Z} \to \mathcal{W}$, $f \circ \mathcal{M} : \mathcal{X}^* \to \mathcal{W}$ satisfies g.*

Lemma 13 ([Dwork and Roth, 2014, Bun and Steinke, 2016])**.** *Every mechanism that satisfies a (ε, δ)-DP or ρ-zCDP guarantee is immune to post-processing.*

The Gaussian mechanism as presented below is one of the most used DP techniques.

Lemma 14 ([Bun and Steinke, 2016, Balle and Wang, 2018], The Gaussian mechanism)**.** *Let $q \colon \mathcal{X}^* \to \mathbb{R}^d$ be a set of queries with ℓ_2 sensitivity $\Delta q := \max_{D \sim D'} \|q(D) - q(D')\|_2$. Then the mechanism that outputs $\tilde{q}(D) = q(D) + Z$ where $Z \sim \mathcal{N}\left(0, (\Delta q)^2 \sigma^2 I_d\right)$ satisfies $(2/\sigma^2)$-zCDP and (ε, δ)-DP for*

$$\delta \geq \Phi\left(\tfrac{1}{2\sigma} - \varepsilon\sigma\right) - e^{\varepsilon}\Phi\left(-\tfrac{1}{2\sigma} - \varepsilon\sigma\right).$$

B More Details of Related Work for Differentially Private Heavy Hitters

Here we present an expanded discussion of the related work on differentially private heavy hitters.

A key technical component of our work is a novel algorithm for the heavy hitters problem for hierarchical data. For this problem, we are given a tree $\mathcal{T}$ with height h. Each data point increase the count of a single leaf by 1, and each internal node stores the sum of counts for all child nodes.[5] We want to privately detect all nodes with count above some threshold τ. Notice that the root of $\mathcal{T}$ is always heavy and the set of heavy hitters form a tree with the same root (except for the special case where no nodes are heavy). We exploit this structure in our algorithm.

[5]We focus on binary trees in the presentation but our algorithm works with any branching factor. More generally, our heavy hitters algorithm is applicable to any setting where adding a data point only changes the value of nodes that are all part of the same path from root to leaf and the value of a parent node is never smaller than any child value.

This problem has connections both to differentially private quantile estimation and sparse histograms. If we restrict our view to a single path from the root to the leaf, we want to find the first node with count below the threshold. This is equivalent to a quantile query (also known as threshold queries). In the (discrete) quantile problem, we have n data points $x_i \in \{1, \dots, h\}$. The goal is to find the largest q such that $|\{i \in [n] : x_i \le q\}| \le \tau$ for some threshold τ. Standard approaches include private selection using the exponential mechanism which has rank error $O(\log h/\varepsilon)$ (or $O(\log h/\sqrt{\rho})$ under ρ-zCDP) [Kaplan et al., 2022] and noisy binary search which has rank error $O(\sqrt{\log h \log\log h/\rho})$ for ρ-zCDP [Huang et al., 2021]. Other techniques achieve better dependence on h (e.g. Bun et al. [2015], Cohen et al. [2023]), but have larger constants and are only preferred for very large h.

If we instead restrict out view to a single layer of the tree, the problem is equivalent to detecting heavy hitters in a histogram. For deeper layers in the tree, this histogram must be sparse, since the number of nodes exceeds the number of data points. In the sparse histogram problem, we have n data points $x_i \in [d]$ for some $d \gg n$. We want to privately estimate $|\{i \in [n] : x_i = j\}|$ for all $j \in [d]$. Variants of this problem have been thoroughly studied in the differential privacy literature [Korolova et al., 2009, Cormode et al., 2012, Gotz et al., 2012, Balcer and Vadhan, 2019, Wilson et al., 2020, Aumüller et al., 2022, Wilkins et al., 2024, Lebeda and Retschmeier, 2025, Kerschbaum et al., 2025]. The optimal error for this problem is $O(\min(\log(d), \log(1/\delta))/\varepsilon)$. We rely on a lower bound from this literature to derive a lower bound for our setting. As such, the heavy hitters problem we consider generalizes both quantile queries and sparse histograms. In fact, we leverage and adapt techniques from both problems in our algorithm.

A related problem is the task of releasing counts for hierarchical data. The straightforward approach samples independent Gaussian noise scaled by $\sqrt{h}$ to each count (here h is the maximum height of a tree and thus the ℓ_2 sensitivity is $\sqrt{h}$). Several papers [Honaker, 2015, Dawson et al., 2023, Ghazi et al., 2026] leverage redundancy in the tree structure to achieve better utility by post-processing the noisy queries. An alternative approach to exploiting redundancy in queries is to directly add correlated noise using the factorization mechanism framework [Li et al., 2015, Edmonds et al., 2020]. Assuming that all nodes on the same layer in $\mathcal{T}$ have the same branching factor, the tree structure can be encoded as h marginal queries. Marginal queries have received significant study in differential privacy and recent results provide optimal factorizations [Xiao et al., 2023, Lebeda et al., 2025, He et al., 2026]. Both the postprocessing techniques and the factorization framework lead to useful constant factor improvements, but the noise still scales with $\sqrt{h}$. Since there are $2^h - 1$ noise samples, one for each node, the expected largest noise sample has magnitude $O_{\varepsilon,\delta}(\sigma\sqrt{\log(2^h)}) = O_{\varepsilon,\delta}(h)$. In many use cases, h is relatively small; for instance, the data used for the US 2020 Decennial Census [Abowd et al., 2022, He et al., 2026] has 6 levels (Country, State, County, Tract, Block Group, and Block), so multiplying noise by a factor $\sqrt{h} = \sqrt{6} \approx 2.45$ is acceptable for these problems. We aim to support significantly deeper trees, and thus require a much lower dependency on the tree height.

Despite a lot of prior work in differential privacy both on heavy hitters estimation and release of hierarchical data, there is limited work on the intersection. Ghazi et al. [2023] consider the same heavy hitters problem as us as part of their work (see Problems 4.2 and 4.4 therein). They reduce the dependency on the height to $O(\sqrt{\log h})$. However, they introduce a $\sqrt{n}$ term, which is typically much higher than $\sqrt{h}$. The most relevant related work is by Biswas et al. [2024] who consider the task of privately releasing Hierarchical Heavy Hitters (HHH), a generalization of the heavy hitters problem. In the HHH problem, the count of a hierarchical heavy hitter is not included in the sum for the parent node. Note that this is a strictly harder problem, since we can recover the tree of heavy hitters by post-processing the HHH set. Biswas et al. [2024] study the problem in both the static and the streaming settings. In the static setting they propose an algorithm inspired by the Sparse Vector Technique (SVT) [Dwork and Roth, 2014, Lyu et al., 2016] with noise scale independent of the height of the tree. The error scales only with $O(\max(\log(nh), \log(h/\delta))/\varepsilon)$. Since their setting is strictly harder than our heavy hitters problem, we would ideally use their algorithm for detecting heavy hitter. Unfortunately, we have discovered a subtle mistake in their privacy proof, as we discuss in Appendix D. The authors recently uploaded a revised version of their paper to arXiv, in which they correct the mistake [Biswas et al., 2026]. The error of their corrected mechanism is $\tilde{O}(\log(1/\delta)\log(h/\delta)/\varepsilon)$, which incurs an additional $O(1/\delta)$ factor compared to their initial result. Consequently, their bound is worse than ours in terms of both h and δ. Moreover, their technique has larger constants than ours, as shown in Section 4.1. Nevertheless, the problem they consider is strictly

more general, and determining whether their upper bound can be improved remains an interesting open question.

Two recent papers [Bernardini et al., 2025, Guo et al., 2026] study the problem of privately identifying frequent substrings. Their techniques relies on a tree-based data structure. Although the setting is not directly comparable to the problem we consider in this paper, we believe (adaptations of) our algorithm may be relevant for identifying frequent substrings under approximate DP.

C Additional Experimental Results and Details

Results on additional datasets. Figure 4 shows results for datasets that were left out of the main text. Similar to Figure 3, we outperform all the private competitors.

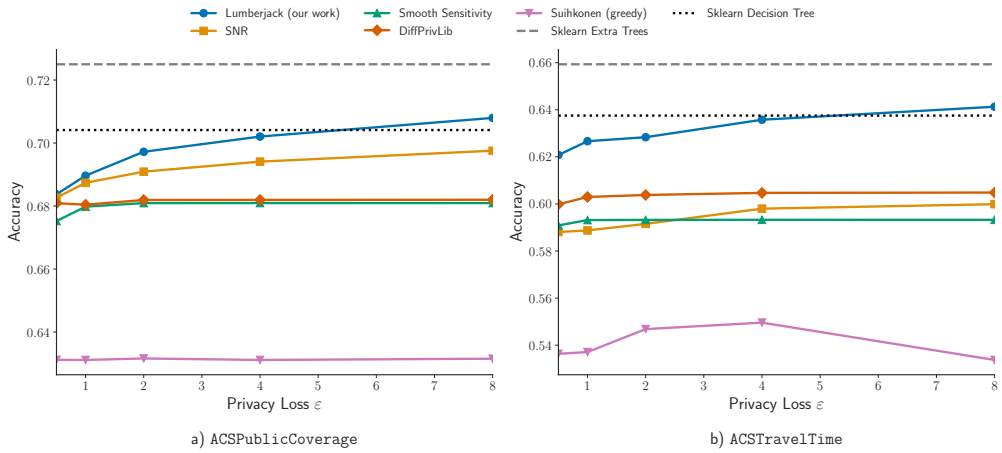

Figure 4: Experimental results for two additional Folktables classification tasks.

Hyperparameters for baselines. The `SNR` implementation computes most hyperparameters based on the input. They do not support continuous features directly, so the domain has to be discretized. We set the number of bins to 5 reflecting parameters used in the paper [Fletcher and Islam, 2015b].

For `Smooth Sensitivity`, we only set the number of trees since the depth is computed using the number of features. We train 30 trees for all benchmarks because they used 30 trees for their experiments with `Adult` [Fletcher and Islam, 2017]. Note that we used the original implementation which contains a known privacy bug. The bug is fixed in the `DiffPrivLib` variant of the algorithm, but we include the original source code as a baseline because the implementations handle categorical features in 2 different ways.

For `DiffPrivLib`, we tuned hyperparameters using the validation set. Strictly speaking, one should spend part of the privacy budget for this tuning, since the choice of hyperparameters can reveal information about the dataset (See Papernot and Steinke [2022]). However, we do not include this in the privacy cost to mimic the best case scenario where an analyst picked good hyperparameters. This implementation often predicts the majority class for all data points if we pick bad parameters. We consider forest sizes in increments of 5 trees and explore tree depth up to 19. The typical selected parameters were 10-20 trees and depth 15-17. Note that increasing the search to deeper trees quickly becomes infeasible due to memory usage since `DiffPrivLib`, unlike `Lumberjack`, stores all empty nodes in the tree. For example, at depth 20, there are more than a million nodes in a tree so most of the memory usage is dedicated to nodes that contain no data points. For `DiffPrivLib`, we did not use one-hot encoding for categorical features. The implementation does not support weighted feature sampling, and the classifier performs much worse with one-hot encoding. Instead, we set the value to the index in the order of the original data.

For `Suihkonen`, we used the same hyperparameters that the original work used for `Adult`, which is a forest of 15 trees with depth 7. When splitting on categorical featuresn they split on a set rather than a single category. They use a parameter to determine the size of subsets, which we set to

8. The accuracy on `Adult` roughly matches the plot from [Suihkonen, 2023]. We did not change the hyperparameters for the `Folktables` tasks except for only considering "one vs. all" splits on categorical attributes. This was required for runtime purposes. The choice of this parameter affects the performance on these datasets, but extensive hyperparameter tuning for this algorithm was impractical since certain choices are computationally expensive. Note that we did not include the implementation of Suihkonen [2023] for ACSEmployment because the runtime was too high for this dataset.

We found privacy concerns in all papers from the greedy decision tree paradigm (see Appendix E), making it unclear which technique should be regarded as the prior state of the art. We use `Suihkonen` as a representative technique from this paradigm. As it performs very poorly on folktables experiments, we considered including an additional baseline based on greedy decision trees. However, we did not find any suitable candidate among existing literature. Early works such as Patil and Singh [2014], Fletcher and Islam [2015a] are known to perform poorly in general, and they do not directly support continuous features. The runtime of Consul and William [2021] is exponential in the size of categorical features, which is infeasible for several of the datasets we consider. Many of the other papers do not provide any source code.

Toy Example in Figure 1. The dataset is constructed by combining the `make_moons` and `make_blobs` dataset generators from scikit-learn. We create 10000 data points in total, 5500 red points, 4000 blue points, and 500 green points. We add an offset to the moons such that the feature bounds are non-negative. The center of the green outlier blob is distance 1 from the upper bound of both features. We set the bounds to $[0, 5]$, $[0, 50]$, and $[0, 5000]$ for Feature 1 and $[0, 4]$, $[0, 40]$, and $[0, 4000]$ for Feature 2. We train 25 trees and use majority voting for classification. We tested various depths for the fully random trees for the $[0, 5] \times [0, 4]$ setting and found that depth 10 splits the classes fairly well. When the bounds increase, most of the feature space is empty. As a result, many of the random splits create empty nodes, and the fully random trees completely fail to separate the moons. For `Lumberjack`, we set $\varepsilon_1 = 0.75\varepsilon$ and use a max depth of 100. Although many of the early splits in each trees create empty leaves for the extreme $[0, 5000] \times [0, 4000]$ setting, our algorithm eventually locates the relevant part of the feature space because it adapts to the number of data points.

D Technical Issue with SVT-based Argument by Biswas et al. [2024]

In this section, we describe a subtle mistake that we discovered in the privacy analysis of an algorithm designed by Biswas et al. [2024] to find Hierarchical Heavy Hitters. At a high level, their algorithm traverses the tree in bottom-up order and privately compares the count of each node against a noisy threshold. Crucially, the noise for this comparison does not scale with h as the algorithm use a technique inspired by the popular Sparse-Vector-Technique (SVT) [Dwork and Roth, 2014, Lyu et al., 2016]. Biswas et al. [2024] consider a more general version of the problem that we address in Section 4, their technique could in principle be used to implement a variant of `Lumberjack`. However, as we show below, the strong privacy properties of SVT do not appear to transfer directly to trees. Despite our efforts, we have not been able to resolve the issue with their algorithm (apart from the trivial fix of increasing the noise scale by a factor $O(\sqrt{h})$).

At a high level, the proof of the (ε, δ)-DP privacy guarantee for Algorithm 1 of the paper [Biswas et al., 2024, Theorem 3.1] proceeds as follows for any two neighboring datasets X and $X' = X \cup x'$: Separate the nodes of the tree into (i) the path that is affected by adding or removing the element x' and (ii) the remaining part of the tree. For the former, they apply an argument similar to the proof of the Sparse-Vector-Technique (SVT) [Dwork and Roth, 2014, Lyu et al., 2016]. Since they add a noise sample to the threshold which *correlates* all these queries, they can output any number of "negative" comparisons ($\perp$) on the path at no additional privacy cost. They ensure that each data point is only part in one "positive" query ($\top$), which ultimately achieves a noise scale that is independent of the height of the tree. For the latter, they point out that the output distributions for these nodes are unaffected by the change in datasets.

We argue that each analysis on it's own is correct, but the two arguments are not compatible and therefore break when combined. On a high level the issue is that the first argument utilizes the fact that the entire output sequence of their algorithm is slightly correlated to achieve stronger privacy, while the second argument relies on parallel composition which requires separate parts of the output to be completely independent. Thus we cannot benefit from the first privacy properties of SVT while

simultaneously ignoring the effect on the other part of the output. We point out concrete parts of the proof that breaks in more details below.

The proof in the paper divides the nodes into three groups, but, to point out the issue in the proof, it is sufficiently to only consider the partition into the affected ("Active" nodes in the paper) and the unaffected ("Unrelated" and "After" nodes in the paper) nodes. Equation 9 in the paper then writes the probability of observing any output $\vec{a} = (a_1, \ldots, a_m)$ as

$$\prod_{i=1}^{m} p(\beta_i, a_i) = \prod_{i \in I_{\text{affected}}} p(\beta_i, a_i) \cdot \prod_{i \in I_{\text{unaffected}}} p(\beta_i, a_i)\,,$$

where m is the number of nodes in the tree, β_i is the random variable of the value of node i in the output of the mechanism using input X and a_i is the observed output. $p(\beta_i, a_i)$ is shorthand notation for $\Pr\left[\beta_i = a_i | \vec{\beta}_{-i} = \vec{a}_{-i}\right]$ where $\vec{\beta}_{-i} = (\beta_1, \ldots, \beta_{i-1})$ and $\vec{a}_{-i} = (a_1, \ldots, a_{i-1})$. The nodes are indexed in a bottom-up level order, and β_i takes on the value $\perp$ or $\top$. β_i' will denote the corresponding random variable when the mechanism is run on X'.

For the unaffected nodes, the proof in [Biswas et al., 2024] states (page 20 and 21, paragraphs *Unrelated Nodes* and *After Nodes*) that for any $i \in I_{\text{unaffected}}$

$$p(\beta_i, a_i) = p(\beta_i', a_i).$$

However, $\vec{\beta}_{-i}$ can also include β_j with $j \in I_{\text{affected}}$. This means that even though β_i and β_i' would have the same distribution unconditionally ($\Pr[\beta_i = a_i] = \Pr[\beta_i' = a_i]$), the equality does not hold, because the conditions of the probabilities are not identical. The noise sample added to the threshold is denoted by γ, and we have $\Pr[\beta_j = a_j|\gamma] \neq \Pr[\beta_j' = a_j|\gamma]$ for any $j \in I_{\text{affected}}$. The value of γ affects all queries and observing β_j and β_j' reveal different information about γ which breaks the equality.

Regarding the affected nodes, the proof applies an SVT-style argument (page 22 and 23, paragraph *Active Nodes*), in which an essential part is conditioning on the threshold noise γ [Lyu et al., 2016]. The proof of Lyu et al. [2016] leverages the fact that the noisy threshold is never revealed to avoid the cost of composition. Unfortunately, in this setting, conditioning on γ in this way is not as straightforward, since the conditions can be affected by observations β_j with $j \in I_{\text{unaffected}}$. Intuitively, we learn information about γ from all the outputs of the unaffected nodes. In the SVT argument, we do not pay the privacy cost for additional queries that return $\perp$. However, queries for $I_{\text{unaffected}}$ can contain both $\perp$ and $\top$, so we cannot ignore their impact in the privacy argument. In extreme cases for huge worst-case input we could infer the approximate value of γ from just the unaffected queries.

It becomes clear that both directions of the privacy proof break due to the shared dependence on γ. It is not possible to index the nodes in a way such that this does not occur, which shows the incompatibility between the two arguments.

E Privacy Concerns in Prior Work on Differentially Private Random Forests

We identified privacy concerns in many published papers on differentially private random forests. In this section, we summarize a selection of these issues. Our goal is not to provide a comprehensive review of all privacy proofs in the literature; rather, we aim to highlight recurring pitfalls. While differential privacy algorithms and implementations are known to be error-prone in general (see, e.g., [Casacuberta et al., 2022, Cebere et al., 2026]), this line of work nonetheless concentrates a number of mistakes that should be readily apparent to experts in differential privacy. Some of these issues directly affect the core claims of the corresponding works, while others could likely be resolved through relatively minor modifications to the algorithms. We do not attempt to categorize the severity of the identified problems. We also note that several papers are written in a way that makes their technical claims difficult to verify. We hope that this discussion will help improve the rigor, reproducibility, and comparability of future work in this area.

An additional purpose of this section is to justify our decision not to include several prior works as baselines in Section 6. Due to the identified privacy concerns, it is unclear which papers should be regarded as the prior state of the art for differentially private random forests. In particular, we found at least one privacy issue in every technique based on greedy random forests. Nevertheless,

our method matches or exceeds the performance claimed by these works (e.g., those using Adult typically report an accuracy of around 79-82%).

Recurring problems. Several types of differential privacy violations appear repeatedly throughout this line of work. One common issue concerns the stopping criterion used during tree construction. In non-private decision trees, it is standard practice to stop splitting when all data points in a node share the same label. Under differential privacy, however, this condition cannot be checked deterministically, since doing so directly reveals information about the underlying data. Unfortunately, we found this issue to be widespread in the literature [Patil and Singh, 2014, Xin et al., 2019, Hou et al., 2019, Guan et al., 2020, Zhang et al., 2021, Liu et al., 2023, Vos et al., 2023, Suihkonen, 2023].

Another recurring problem arises in methods that use the exponential mechanism to select split points for continuous features. The candidate split set itself must be chosen in a privacy-preserving manner. In particular, using the observed feature values directly as the only candidate split points reveals information about the exact locations of data points and therefore violates differential privacy. One possible remedy is to sample intervals between adjacent data points proportionally to their length and utility, and then select a point uniformly within the chosen interval, as proposed by Kaplan et al. [2022]. Nevertheless, several works instead construct a data-dependent discrete candidate set, which violates differential privacy [Li and Li, 2017, Guan et al., 2020].

Finally, Patil and Singh [2014], Li and Li [2017] employ bootstrapping without properly accounting for its privacy cost, resulting in incorrect privacy guarantees.

Specific issues in each paper. Below we point to the location of specific issues in each paper.

1. Patil and Singh [2014] deterministically stop splitting when all samples in a node belong to the same class (Section IV, subsection D, point 2). This data-dependent control flow in the construction of the tree violates DP. Additionally, they do not properly account for bootstrapping which results in an inaccurate privacy guarantee (Algorithm 1, step 3).
2. Fletcher and Islam [2015a] calculate the local sensitivity based on noisy values instead of using data-independent global sensitivity (Algorithm 1, line 17).
3. Li and Li [2017] use the midpoint between adjacent data points as candidates when splitting continuous features (Section III, Subsection D, Equation 5). This is heuristically better than using the data points themselves, but it still breaks DP. They also do not properly account for bootstrapping (Step 2a of the algorithm in Section III, Subsection B).
4. Hou et al. [2019] use a deterministic data-dependent stopping criterion in the splitting algorithm (Algorithm 1, Line 5). Quoting directly from the paper: "The condition that the decision tree stops growing includes: 1) All samples in the node have the same classification result".
5. Xin et al. [2019] have data-dependent stopping conditions in their tree-building process. They do not split nodes with 10 or fewer data points (Algorithm 2, Line 1), and they deterministically stop splitting pure nodes (Algorithm 2, line 5).
6. Guan et al. [2020] sort values of continuous attributes and use the average of buckets with 5 data points as candidates for the splitpoint (Algorithm 1, Line 10). The candidate set differs between neighboring datasets which breaks DP. Their implementation also deterministically does not split on pure nodes (Algorithm 1, line 4).
7. Zhang et al. [2021] deterministically stop splitting when all samples in a node belong to the same class (Algorithms 1, 2, and 3, see termination condition).
8. Consul and William [2021] use data-dependent sensitivity in both the splits (Equation 6) and the leaf nodes in the case of regression trees (Section 3, paragraph *Estimating leaf node parameters*).[6]
9. Liu et al. [2023] deterministically stop splitting on pure nodes (Algorithm 2, Line 4). In the second part of their algorithm they sample points with weights based on misclassification without accounting for it (Algorithm 3, Line 9). This leads to a higher privacy loss for outlier data points.

[6]Incidentally, we note that we could not use this algorithm as a baseline for computational reasons. They inspect all possible subsets when splitting categorical features. This is infeasible for features with many values. The native-country feature from the Adult dataset has 41 possible values so they should evaluate 2^{40} possible splits.

10. Vos et al. [2023] (who focus on the setting of a single decision tree rather than building a forest) deterministically stop splitting at pure nodes (Algorithm 1, line 7).
11. While we use Suihkonen [2023] as a baseline for greedy forests in our experiments, the code does contain some privacy violations including data-dependent stopping criteria for splitting (line 496 in the implementation), deterministically labeling pure leaf nodes (line 492 in the implementation) and sensitivity miscalibration (line 242 in the implementation).

Finally, we note that Fletcher and Islam [2017] originally contained a privacy error that was later corrected. The original algorithm used a variant of the exponential mechanism calibrated with *smooth sensitivity*. Their random forest method was subsequently implemented in the popular open-source library DiffPrivLib [Holohan et al., 2019]. During the development, Naoise Holohan identified a mistake in the privacy proof for the smooth sensitivity approach. As a result, the DiffPrivLib implementation was updated to use the Permute-and-Flip algorithm [McKenna and Sheldon, 2020] instead, and Fletcher & Islam later added an addendum to an updated arXiv version of the paper describing the issue. We commend Fletcher & Islam, and the DiffPrivLib developers, for publicly documenting and correcting the issue.

F Missing Proofs from Main Body

Here we present proofs of technical results from the main body of the paper.

Proof of Lemma 3.

Proof. The element x is associated with a unique path P_x from the root to a leaf. We want to show that at most $1 + \lfloor \log_2(h) \rfloor$ nodes along P_x are queried. In Phase 1, the algorithm queries one node $u \in P_x$ at level $\text{mid} = \lfloor h/2 \rfloor$ (if it is not already marked). We show in the following that P_x is queried in at most one recursive call of the algorithm, which implies the desired bound. The mark of u determines which recursive phase may subsequently query nodes on P_x:

- **Case 1: u is marked *Light*.** All descendants of u (the bottom half of P_x) are marked *Light*. Consequently, the recursive step containing P_x is skipped in Phase 2 by the condition on Line 11. Any further queries to P_x occur in the recursive call on the top part of the tree in Phase 3.
- **Case 2: u is marked *Heavy*.** All ancestors of u (the top half of P_x) are marked *Heavy*. Because marked nodes are skipped by Line 2, Phase 3 never queries any nodes in P_x. When $h > 2$, P_x is part of exactly one recursive call in Phase 2 via the subtree $\mathcal{T}_v$ that contains P_x.

In both cases, x is involved in exactly one query and at most one recursive call on a tree of height $\leq \lfloor h/2 \rfloor$. Let $Q(h)$ be the maximum number of queries for x in a tree of height h. We have the recurrence: $Q(h) \leq 1 + Q(\lfloor h/2 \rfloor)$. With the base case $Q(1) = 1$, this yields $Q(h) \leq 1 + \lfloor \log_2(h) \rfloor$ as desired. □

Proof of Lemma 4.

Proof. It follows from the monotonicity of the tree structure. The statement clearly holds for any nodes that are queried by CHECKTHRESHOLD. We have to show that it also holds for nodes that are marked without being directly queried. Such nodes that are marked *Light* are descendants of some node u where we had $\text{CHECKTHRESHOLD}_\tau(\#count(u)) = \perp$. These nodes therefore have a count $< \tau + \beta$ since u cannot have a lower count than any of it's descendant. Similarly, nodes that are marked *Heavy* are ancestors of some node where CHECKTHRESHOLD returned $\top$ and thus must have a count of $> \tau - \alpha$. □

Proof of Lemma 5.

Proof. Since $0 \leq \tau - \Delta - 1$ all empty nodes are correctly classified as *Light*. Let m denote the number of calls to CHECKTHRESHOLD with non-zero counts performed by Algorithm 1. It follows from Lemma 3 that $m \leq n\lceil \log_2(h+1) \rceil$. Let $(Z_1, \ldots, Z_m)$ denote the noise samples for these

queries. By a standard Gaussian tail bound we have $\Pr[|Z_i| \geq t \cdot \sigma] \leq 2\exp(-t^2/2)$ and thus $\Pr[|Z_i| \geq \alpha] \leq \beta/m$. By applying a union bound we thus have $\Pr[\max_{i\in[m]} |Z_i| \geq \alpha] \leq \beta$. □

Proof of Lemma 7.

Proof. Here we use the looser add-the-deltas approach discussed in Wilkins et al. [2024]. The Gaussian noise is calibrated to satisfy $(\varepsilon, \delta/2)$-DP for ℓ_2-sensitivity $\sqrt{m} = \sqrt{k(1 + \lfloor \log_2 h \rfloor)}$ using [Dwork and Roth, 2014, Appendix A]. We have that $\Pr[\mathcal{N}(0, \sigma^2) \geq \Delta] \leq \delta/(2m)$ which implies that the maximum noise for m queries is less than Δ with probability at least $1 - \delta/2$. □

G Privacy Guarantees of Algorithm 2

Algorithm 2 is an adaptive variant of the Gaussian Sparse Histogram Mechanism from [Wilkins et al., 2024]. The privacy properties from their work apply to this adaptive setting when the number of queries affected by any data point is bounded.

Lemma 15. *Let $(q_1, \ldots, q_d)$ be (adaptively chosen) counting queries such that for any $x \in \mathcal{X}$, we have $q_i(x) = 1$ for at most m values of $i \in [d]$ and $q_j(x) = 0$ for all other queries. Then Algorithm 2 with parameters σ and Δ satisfies (ε, δ)-differential privacy where $\gamma(j) = (m - j) \log \Phi\left(\frac{\Delta}{\sigma}\right)$ and*

$$\delta \geq \max\Bigg[1 - \Phi\left(\frac{\Delta}{\sigma}\right)^m,$$
$$\max_{j\in[m]} 1 - \Phi\left(\frac{\Delta}{\sigma}\right)^{m-j} + \Phi\left(\frac{\Delta}{\sigma}\right)^{m-j}\left[\Phi\left(\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon - \gamma(j))\sigma}{\sqrt{j}}\right) - e^{\varepsilon-\gamma(j)}\Phi\left(-\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon - \gamma(j))\,\sigma}{\sqrt{j}}\right)\right],$$
$$\max_{j\in[m]} \Phi\left(\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon + \gamma(j))\,\sigma}{\sqrt{j}}\right) - e^{\varepsilon+\gamma(j)}\Phi\left(-\frac{\sqrt{j}}{2\sigma} - \frac{(\varepsilon + \gamma(j))\sigma}{\sqrt{j}}\right)\Bigg].$$

Proof. It follows from the proof of [Wilkins et al., 2024, Theorem 5.4]. Wilkins et al. [2024] study an algorithm in the non-adaptive setting, but the algorithm does not rely on any correlation between queries so the same analysis can be applied for adaptive queries. The privacy guarantees follow from standard composition properties of the Gaussian mechanism combined with a bound of the probability of infinite privacy loss events for queries that have a value $< \tau - \Delta - 1$ for one dataset and $\geq \tau - \Delta - 1$ for the other. The proof implicitly requires that the queries are monotonic. That is, all d queries should be either non-increasing or non-decreasing between neighboring datasets. This assumption clearly holds for our setting since we, like [Wilkins et al., 2024], study counting queries. □

H Deterministic Error Bound for Heavy Hitter Algorithm

Here we present a variant of our heavy hitters algorithm with deterministic error bound. We modify Algorithm 2 to always output $\top$ for queries that are sufficiently above the threshold. We use the add-the-deltas technique discussed in Wilkins et al. [2024] to analyze the privacy guarantees.

Algorithm 3 GaussianSparseTwosidedThreshold (GSTT)

Require: Parameters σ, τ, and Δ.
Require: Dataset D, (adaptive) counting queries $q_1, q_2, q_3, \ldots$
for all $i \in \{1, 2, 3, \ldots\}$ **do**
  Sample $Z_i \sim \mathcal{N}\left(0, \sigma^2\right)$.
  if $q_i(D) \geq \tau + \Delta + 1$ or ($q_i(D) > \tau - \Delta - 1$ and $q_i(D) + Z_i > \tau$) **then**
    Release $\top$.
  else
    Release $\perp$.
  end if
end for

Lemma 16. *Let $(q_1, \dots, q_d)$ be (adaptively chosen) counting queries such that for any $x \in \mathcal{X}$, we have $q_i(x) = 1$ for at most m values of $i \in [d]$ and $q_j(x) = 0$ for all other queries. Then Algorithm 3 with parameters σ and Δ satisfies $(\varepsilon, \delta_{Gauss} + \delta_{inf})$-differential privacy where*

$$\delta_{Gauss} = \Phi\left(\frac{\sqrt{m}}{2\sigma} - \frac{\varepsilon\sigma}{\sqrt{m}}\right) - e^{\varepsilon}\Phi\left(-\frac{\sqrt{m}}{2\sigma} - \frac{\varepsilon\sigma}{\sqrt{m}}\right), \quad \textit{and} \quad \delta_{inf} = 1 - \Phi\left(\frac{\Delta}{\sigma}\right)^m.$$

Proof. By Definition 1, we have to show that for any pair of neighboring datasets D and D' and all sets of outputs Z we have

$$\Pr[\mathrm{GSTT}(q_1(D), \dots, q_d(D)) \in Z] \le e^{\varepsilon}\Pr[\mathrm{GSTT}(q_1(D'), \dots, q_d(D')) \in Z] + \delta_{\text{Gauss}} + \delta_{\text{inf}}.$$

We prove that the inequality above by introducing a series of values $(\hat{q}_1, \dots, \hat{q}_d)$ that "lies between" $(q_1(D), \dots, q_d(D))$ and $(q_1(D'), \dots, q_d(D'))$. Consider the set of all possible counting queries $\mathcal{Q}$. Let $S \subseteq \mathcal{Q}$ denote the set of counting queries that are *not* classified deterministically for D. That is, $q(D) \in [\tau - \Delta - 1, \tau + \Delta + 1]$ if and only if $q \in S$ and define similarly S' and $\hat{S}$. We construct the intermediate values such that $S' = \hat{S}$. For all $q_i \in (S \cap S')$ we set $\hat{q}_i = q_i(D)$ and for all $q_i \notin (S \cap S')$ we set $\hat{q}_i = q_i(D')$. In other words, we construct $(\hat{q}_1, \dots, \hat{q}_d)$ such that (1) $(q_1(D'), \dots, q_d(D'))$ and $(\hat{q}_1, \dots, \hat{q}_d)$ only differ in entries that are in both S and S' (2) $(q_1(D), \dots, q_d(D))$ and $(\hat{q}_1, \dots, \hat{q}_d)$ only differ in entries that are in only one of S and S'. This allows us to analyze each case separately to derive our values for δ_{Gauss} and δ_{inf}. Note that the values $(\hat{q}_1, \dots, \hat{q}_d)$ might not correspond to evaluating counting queries on any existing dataset $\hat{D}$. We just use the values only for the proof and utilize the fact that Algorithm 3 only access the count. Notice that we cannot have any query where $q_i(D) < \tau - \Delta - 1$ and $q_i(D') > \tau + \Delta + 1$ or vice versa. As such, whenever $q_i \notin (S \cup S')$ the deterministic output of Algorithm 3 is the same for both datasets. Therefore we only have to consider queries that are deterministic for exactly one or for none of the datasets. The intermediate values allow us to analyze the two cases separately.

First we show that

$$\Pr[\mathrm{GSTT}(q_1(D), \dots, q_d(D)) \in Z] \le \Pr[\mathrm{GSTT}(\hat{q}_1, \dots, \hat{q}_d) \in Z] + \delta_{\text{inf}}.$$

Since $q_i(D) \neq \hat{q}_i$ for at most m queries any change in output distribution is due to those queries. For each such query, one of the outputs is deterministic. Without loss of generality, assume that $q_i(D) < \tau - \Delta - 1$ such that the output is always $\perp$. The probability that the output is $\perp$ with $\hat{q}_i$ is at least $\Phi\left(\frac{\Delta}{\sigma}\right)$, since Z_i must be larger than Δ to change the output to $\top$. The probability that all the m queries are identical is therefore at least $\Phi\left(\frac{\Delta}{\sigma}\right)^m$ which gives the bound for δ_{inf}.

Next consider the case of $(\hat{q}_1, \dots, \hat{q}_d)$ and $(q_1(D'), \dots, q_d(D'))$. These inputs only differ for queries in $[\tau - \Delta - 1, \tau + \Delta + 1]$. There are at most m differing queries, so the ℓ_2-distance is bound by $\sqrt{m}$. The algorithm can be seen as running the Gaussian mechanism on values in $[\tau - \Delta - 1, \tau + \Delta + 1]$ and post-processing the values. The tight bound for the Gaussian mechanism (Lemma 14) gives us

$$\Pr[\mathrm{GSTT}(\hat{q}_1, \dots, \hat{q}_d) \in Z] \le e^{\varepsilon}\Pr[\mathrm{GSTT}(q_1(D'), \dots, q_d(D')) \in Z] + \delta_{\text{Gauss}}.$$

Note that we technically cannot apply Lemma 14 directly, since the Gaussian mechanism is typically analyzed in the static setting, while we allow adaptively chosen queries. A more rigorous proof would fix the neighboring datasets and study the input as a series of adaptive queries rather than a fixed sequence. Under Gaussian Differential Privacy [Dong et al., 2022] we (informally) allow an analyst to make m queries that are each $1/\sigma$-GDP and an unlimited number of 0-GDP queries (those where the counts are the same in our original setup). Using the adaptive composition property and conversion from μ-GDP to (ε, δ)-DP recovers the inequality above. We leave the details as an exercise for the reader.

Combining the two inequalities above give us the desired bound

$$\begin{aligned}\Pr[\mathrm{GSTT}(q_1(D), \dots, q_d(D)) \in Z] &\le \Pr[\mathrm{GSTT}(\hat{q}_1, \dots, \hat{q}_d) \in Z] + \delta_{\text{inf}} \\ &\le e^{\varepsilon}\Pr[\mathrm{GSTT}(q_1(D'), \dots, q_d(D')) \in Z] + \delta_{\text{Gauss}} + \delta_{\text{inf}}.\end{aligned}$$

□

Similarly to Lemma 7, we give a simple upper bound for the values of σ and Δ.

Corollary 17. *When $\varepsilon < 1$ then the parameters*

$$\sigma = \frac{\sqrt{2m\ln(2.5/\delta)}}{\varepsilon} \qquad \Delta = \sigma \cdot \sqrt{2\ln(2m/\delta)} = O\left(\frac{\sqrt{m}\log(1/\delta)}{\varepsilon}\right)$$

satisfy the condition in Lemma 16 for (ε, δ)-DP. In the last equality we assume that $m \ll 1/\delta$.

Since Algorithm 3 clearly can only misclassify queries in the interval $[\tau - \Delta - 1, \tau + \Delta + 1]$, it follows that we achieve a deterministic error bound for the heavy hitters problem.

Corollary 18. *Let $\mathcal{A}$ denote an algorithm that takes as input k trees with maximum height h. The algorithm runs Algorithm 1 for each tree where* CHECKTHRESHOLD *is instantiated as Algorithm 3 with parameters σ, τ, and Δ. Then $\mathcal{A}$ satisfies (ε, δ)-DP with values specified by Lemma 16. Let $\alpha \in \mathbb{R}$ denote the smallest value such that all nodes with at least count $\tau + \alpha$ are marked as Heavy and all nodes with at most count $\tau - \alpha$ are marked Light. Then it holds with probability 1 that $\alpha \leq 1 + \Delta$. Furthermore, when $\tau \geq 1 + \Delta$ we can implement the algorithm to run in time and space $O(knh)$.*

We note that it is possible to get tighter results using a case by case analysis similar to Lemma 15.

I Proof of Lower Bound (Lemma 8)

Here we discuss the lower bound for deep trees presented in Lemma 8. We use the lower bound of Balcer and Vadhan [2019] which we restate below. Note that Balcer and Vadhan use a different definition of differential privacy. They study the problem under the replacement definition for neighboring datasets where the size of the dataset is known (bounded DP) while we use the add/remove (unbounded DP) definition. However, we can still use their result since any DP mechanism under add/remove satisfies DP with slightly worse parameters under bounded DP by a simple group privacy argument.

Lemma 19. *A heavy hitters algorithm $\mathcal{M}$ that satisfies (ε, δ)-DP under the add/remove neighboring relation satisfies $(2\varepsilon, (1 + e^{\varepsilon})\delta)$-DP under the replacement definition.*

Proof. This is well known in differential privacy. We include a short proof for completeness. Let i be the index that differs between some pair of neighboring datasets $D \sim D'$ under replacement. That is, $D_j = D'_j$ for all $j \neq i$. Then define an intermediate dataset $\hat{D} = D \setminus \{D_i\} = D' \setminus \{D'_i\}$. We have

$$\begin{aligned}
\Pr[\mathcal{M}(D) \in Z] &\leq e^{\varepsilon} \Pr[\mathcal{M}(\hat{D}) \in Z] + \delta \\
&\leq e^{\varepsilon}(e^{\varepsilon} \Pr[\mathcal{M}(D') \in Z] + \delta) + \delta \\
&= e^{2\varepsilon} \Pr[\mathcal{M}(D') \in Z] + (1 + e^{\varepsilon})\delta\,,
\end{aligned}$$

where the inequalities follow from the fact that $D \sim \hat{D}$ and $\hat{D} \sim D'$ are pairs of neighboring datasets under the add/remove definition for which $\mathcal{M}$ satisfies (ε, δ)-DP. □

Lemma 20 (Theorem 7.2 of Balcer and Vadhan [2019])**.** *Let $\mathcal{M}$ be a (ε, δ)-differentially private algorithm (under bounded DP) that takes as input any dataset of size n and d counting queries $(q_1(D), \ldots, q_d(D))$ where each data point satisfies exactly one counting query. Assume that $\mathcal{M}$ has the following error guarantee*

$$\forall D \in \mathcal{X}^n \quad \forall i \in [d] \quad \Pr[|\mathcal{M}(D)_i - q_i(D)| \leq \alpha] \geq 1 - \beta$$

with $\beta \in (0, 1/2]$. If $\mathcal{M}$ outputs at most $\hat{n}$ non-empty bins. Then

$$\alpha \geq \frac{1}{2} \cdot \min\left(\frac{1}{2\varepsilon}\ln\left(\frac{d}{16\beta\hat{n}}\right) - 1, \frac{1}{\varepsilon}\ln\left(\frac{\varepsilon}{4\delta}\right) - 1, n\right)\,.$$

Balcer and Vadhan refer to the type of error guarantee from the above lemma as the (α, β)-Per-Query error. We use the following statement in the proof.

Lemma 21 ([Balcer and Vadhan, 2019, Figure 3])**.** *The mechanism that adds Geometric noise to all counting queries satisfies (α, β)-Per-Query accuracy (under bounded DP) for any*

$$\alpha \geq \lceil 2/\varepsilon \cdot \ln(1/\beta) \rceil\,.$$

Now we restate an exact version of our informal lower bound Lemma 8. Note that we do not optimize constants. For simplicity, we state the result only for $\varepsilon < 1$. This restriction is only used to easily deal with the e^{ε} part of the $(1 + e^{\varepsilon})\delta$ term that arises from changing DP definitions. For a tighter analysis, one should directly analyze our mechanism under replacement DP or carry out the lower bound proof of Balcer and Vadhan [2019] under add/remove DP.

Lemma 22. *Let $\mathcal{M}$ denote an (ε, δ)-DP mechanism (for some $\varepsilon < 1$) that takes as input a tree $\mathcal{T}$ of height $h > 6 + \log_2(n^2/\beta^{15})$ and a dataset of size n and outputs a set of at most n^2 nodes per layer of $\mathcal{T}$. Suppose that with probability at least $1 - \beta$, $\mathcal{M}$ outputs all nodes in $\mathcal{T}$ with count above some value $\hat{\tau} \in \mathbb{R}$. Then*

$$\hat{\tau} \geq \frac{1}{2} \cdot \min\left(\frac{1}{8\varepsilon} \ln\left(\frac{2^{h-1}}{32\beta n^2}\right) - 1, \frac{1}{4\varepsilon} \ln\left(\frac{\varepsilon}{4\delta}\right) - 1, n\right).$$

.

Proof. We can encode any sparse histogram (see Lemma 20) with $d = 2^{h-1}$ queries in the leaves of the tree. Label the leaves in arbitrary order and set the count of the first leaf to $q_1(D)$, the count of the second leaf to $q_2(D)$ etc. By Lemma 19, $\mathcal{M}$ satisfies $(2\varepsilon, (1 + e^{\varepsilon})\delta) < (2\varepsilon, 4\delta)$-DP in the setting of Balcer and Vadhan [2019]. We can spend additional privacy budget to add Geometric noise calibrated for $(2\varepsilon, 0)$-DP to the all leaves of $\mathcal{T}$ that are part of the output of $\mathcal{M}$. This clearly does not violate the requirement of outputting at most $\hat{n} = n^2$ non-empty bins. For any leaf that was part of the output of $\mathcal{M}$, the (α, β) accuracy is $\lceil 1/\varepsilon \cdot \ln(1/\beta) \rceil$ by Lemma 21. Thus all queries above $\hat{\tau}$ have this error with failure probability at most 2β. By our assumption on h the error of nodes above $\hat{\tau}$ is lower than the error required by the lower bound. The leaves with value below $\hat{\tau}$ must therefore increase the (α, β)-Per-Query error. We have no requirements on $\mathcal{M}$ for nodes with count less than $\hat{\tau}$. If we simply output an estimate of zero for all nodes not in the output of $\mathcal{M}$, these leaves can therefore have expected error up to $\hat{\tau}$. Since the mechanism above is $(4\varepsilon, 4\delta)$-DP and has failure probability at most 2β by Lemma 20 we must have

$$\hat{\tau} \geq \frac{1}{2} \cdot \min\left(\frac{1}{8\varepsilon} \ln\left(\frac{d}{32\beta n^2}\right) - 1, \frac{1}{4\varepsilon} \ln\left(\frac{\varepsilon}{4\delta}\right) - 1, n\right).$$

□

J Implementation Details

Tree construction. In this section we present implementation details of our algorithm. In particular, we discuss how to implement Algorithm 1 to run in time and space $O(\ell)$, where ℓ is the number of nodes with true count at least $\tau - \Delta - 1$. Since each data point increments the count for a single node in each layer of the tree we clearly have $\ell = O(nh)$ as claimed in Theorem 6. Note that in the pseudocode we explicitly mark all nodes either *Heavy* or *Light* which is infeasible for large trees. In practice, we implement the algorithm to return the set of nodes marked *Heavy*. Any nodes that are not explicitly marked by the algorithm are not returned as they are implicitly *Light*.

First, we briefly discuss how we build the unpruned decision tree in time $O(nh)$ starting with the root. We assume that we always select a splitting feature and threshold in time $O(1)$. We then split a node with m points in time $O(m)$ using a simple pass over the nodes to classify them for the left or right child node. Since each data point is part of exactly one split in each layer, the total running time is $O(nh)$. We do not split empty nodes since all nodes in empty subtrees are eventually marked *Light* when $\tau > 1 + \Delta$. If the threshold is larger, we can similarly stop splitting nodes with less than $\tau - \Delta - 1$ data points.

Next, we show that each part of Algorithm 1 can be implemented in linear running time in the size of the tree $O(\ell)$. In Phase 1 we query all nodes in L_{mid} and mark ancestors or descendants. Marking ancestor of a node takes up to time $O(h)$. However, we stop early when we encounter a node which was already marked, so in total over the entire run of the algorithm this takes at most time $O(\ell)$. We do not actually need to explicitly mark all descendants of *Light* nodes, since we skip those recursive calls in Phase 2. After we finish marking nodes in the tree we find the set of heavy hitters by adding nodes starting from the root. Since we stop exploring any path when we encounter a *Light* node, we never explore the unmarked subtrees.

Now, notice that we call CHECKTHRESHOLD$_\tau(\#count(u))$ for any node u at most once for the entire execution of all recursive calls. If we can access all non-zero nodes in L_{mid} in time $O(|L_{\text{mid}}|)$ the entire algorithm therefore runs in $O(\ell)$ (We assume that we sample a Gaussian in time $O(1)$). However, simply traversing the tree from the root to find L_{mid} takes time up to $O(h|L_{\text{mid}}|)$. We reduce the running time with some algorithmic engineering.

We label the nodes (including missing empty nodes) in top-down and left-to-right order. The root has label 1, the layer below the root has labels 2 and 3, the next layer contains labels 4 to 7 etc. For any node with label i, the left child has label $2 \cdot i$ and the right child has label $2 \cdot i + 1$. This is a standard technique for storing a binary tree in an array without using pointers. This labeling is convenient, because when the top node in $\mathcal{T}$ has label i, then L_{mid} are exactly the nodes with labels from $i \cdot 2^{\text{mid}}$ to $(i+1) \cdot 2^{\text{mid}} - 1$, inclusive. However, for large trees most of these nodes do not exist, so we need to efficiently find all such nodes without explicitly querying each label. Ideally, we do not want to use any advanced data structure to avoid adding unnecessary complexity.

Notice that our algorithm always explores the tree in left-to-right order. In particular, if we recursively call the algorithm in ascending label order in Phase 2, then for any particular layer we query the nodes in ascending label order. As such, when we call CHECKTHRESHOLD$_\tau(\#count(u))$ we know that no recursive call will ever need to query CHECKTHRESHOLD$_\tau(\#count(v))$ for any v where u and v has the same depth in the full tree and label$(v) <$ label(u). We utilize this observation to achieve an efficient implementation. We store a queue for each layer in the tree and add all nodes in the tree to the corresponding queue using a single traversal of the tree. When we need to access L_{mid} in a recursive call where the root of the subtree has label i we discard all nodes in the corresponding queue with label $< i \cdot 2^{\text{mid}}$. We then query all nodes from the queue with label $< (i+1) \cdot 2^{\text{mid}}$. Those are exactly the non-empty nodes in L_{mid} and we query them in left-to-right order as desired.

Leaf algorithm. We now briefly discuss the privacy analysis of the two leaf algorithms used in our implementation. The techniques we use are standard. We use the exponential mechanism for majority vote and Gaussian noise for approximate class distributions. We use zCDP for accounting in both cases. For the class distribution we simply add noise from $\mathcal{N}(0, k/(2\rho))$ to each class count. This can create negative counts, but we can clip to non-negative values as post-processing.

Rogers and Steinke [2021] show that the exponential mechanism with privacy parameter ε satisfies $\frac{1}{8}\varepsilon^2$-zCDP by showing it satisfies a condition known as bounded range ε-bounded range. Moreover, in the proof for Lemma 3 in [Rogers and Steinke, 2021], they make use of the bound

$$-\Delta \leq \ell(y, x') - \ell(y, x) \leq \Delta$$

which can be improved in our case, because the scoring function - being the count of each class - is monotonic. Depending on whether we add or remove a row from the dataset, the term $\ell(y, x') - \ell(y, x)$ is in $[0, \Delta]$ or $[-\Delta, 0]$. It follows that the mechanism satisfies $\varepsilon/2$-bounded range and $\frac{1}{32}\varepsilon^2$-zCDP. Monotonicity therefore makes it sufficient to set the ε parameter of the standard exponential mechanism McSherry and Talwar [2007] to $\frac{1}{\sqrt{8\rho/k}}$ instead of $\frac{2}{\sqrt{8\rho/k}}$.

K Regression Trees

In the main paper, we focused on classification problems, in line with prior work. In this section, we discuss how to extend our approach to regression trees. The tree construction algorithm is the same; the only difference is the algorithm used for privatizing leaves. In regression, data points no longer have a discrete label $y \in \mathcal{Y}$, but instead have a numeric one $y \in \mathbb{R}$. We assume that y is bounded to some known interval $[l, u]$. We label each leaf with a private estimate of the label mean of data points in the leaf. Specifically, we first center the queries such that $\hat{y} = y - \frac{u-l}{2}$, and privately estimate the sum of shifted values, whose sensitivity is bounded by $\frac{u-l}{2}$. We then divide the noisy sum by a noisy estimate of the count (clipped to be at least 1), and project the private estimate back to the original range. We split the privacy budget equally between the two queries. This is a standard approach for mean estimation in the add/remove setting; see Algorithm 2 in Kulesza et al. [2024] for more details (note that we use Gaussian noise whereas they use Laplace noise).

We evaluate our mechanism on two datasets. `MedicalCharges` (OpenML ID 45048 `https://www.openml.org/search?type=data&sort=runs&id=45048`) has 163,065 records, 3 numerical features, and the target is the logarithm of the average total medical expenses.

`CaliforniaHousing` [Pace and Barry, 1997] has 20,640 records (one per census block group in the 1990 U.S. census), 8 numerical features, and the target is the median house value in units of $100,000, capped at 5.

We fit 30 trees and use the same hyperparameters as in Section 6, except that we allocate more privacy budget for leaf queries by setting $\varepsilon_1 = 0.25\varepsilon$. We compare against Sklearn Extra Tree Regressor (50 trees) and a Sklearn Decision Tree Regressor. In Figure 5 we report R^2 value and RMSE (Root Mean Square Error). For the Extra Tree models we report results for two splitting threshold. Each experiment is rerun 5 times.

These preliminary results show that our technique supports regression trees. However, the performance is worse than in the classification setting: we need larger privacy budgets to approach the performance of the non-private baselines. One of the key challenges for regression is that the noisy mean has large variance for small leaves, which can significantly affect the aggregate prediction. This could be somewhat mitigated by ensuring that leaves have sufficiently many data points to provide an accurate mean estimate. However, this limits the trees' ability to fit the data. The non-private baseline shows that performance can degrade with higher splitting threshold. This suggests that high utility for regression may require large datasets. We leave the further exploration of private regression forests to future work.

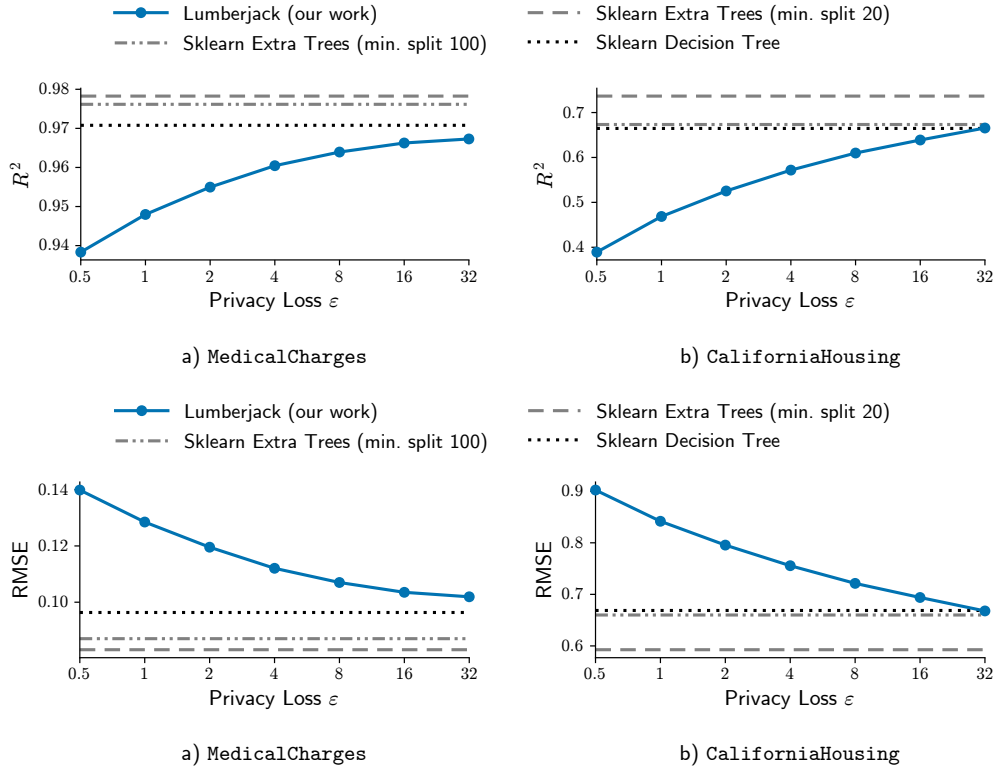

Figure 5: Experimental results for regression datasets. Since prior work does not consider regression we only compare with non-private baselines.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our theoretical contribution for heavy hitters matches the abstract, we show in experiments that we significantly outperform prior work. Additionally, we highlight privacy issues for prior work in Appendix E.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We point out that our heavy hitters algorithm has a minimum requirement for the splitting threshold in Section 4.2. This means that we may be forced to use larger splitting threshold than a non-private implementation which can negatively affects performance at small privacy budgets. This is inherent to the problem under DP, although there is a gap between the upper and lower bound.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: All novel results are backed up by proofs. Most are in the appendix due to space limitations.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We discuss hyperparameters in our experiments section and provide source code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility.

In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code is attached as supplementary material.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: See Section 6.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Each data point is the mean of 5 repetitions.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: Experiments can run on a laptop. The slowest implementation is a baseline. `Smooth Sensitivity` takes 15-20 min for `Adult`.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We introduce a new privacy-preserving model.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We show that it is possible to train better differentially private random forests. We hope that our work inspires more research in this underexplored direction.

    Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use two standard datasets and cite them.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Supplementary material includes instructions for the implementation used for experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: [N/A]

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were used for standard light editing tasks and for assistance with parts of the implementation for the experiments.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.