Federated Boosted Decision Trees with Differential Privacy

Samuel Maddock*
University of Warwick

Graham Cormode
Meta AI

Tianhao Wang†
University of Virginia

Carsten Maple
University of Warwick

Somesh Jha†
University of Wisconsin-Madison

ABSTRACT

There is great demand for scalable, secure, and efficient privacy-preserving machine learning models that can be trained over distributed data. While deep learning models typically achieve the best results in a centralized non-secure setting, different models can excel when privacy and communication constraints are imposed. Instead, tree-based approaches such as XGBoost have attracted much attention for their high performance and ease of use; in particular, they often achieve state-of-the-art results on tabular data. Consequently, several recent works have focused on translating Gradient Boosted Decision Tree (GBDT) models like XGBoost into federated settings, via cryptographic mechanisms such as Homomorphic Encryption (HE) and Secure Multi-Party Computation (MPC). However, these do not always provide formal privacy guarantees, or consider the full range of hyperparameters and implementation settings. In this work, we implement the GBDT model under Differential Privacy (DP). We propose a general framework that captures and extends existing approaches for differentially private decision trees. Our framework of methods is tailored to the federated setting, and we show that with a careful choice of techniques it is possible to achieve very high utility while maintaining strong levels of privacy.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Computing methodologies → Boosting**.

KEYWORDS

Gradient Boosting, Differential Privacy, Federated Learning

ACM Reference Format:
Samuel Maddock, Graham Cormode, Tianhao Wang, Carsten Maple, and Somesh Jha. 2022. Federated Boosted Decision Trees with Differential Privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 23 pages. https://doi.org/10.1145/3548606.3560687

*Author correspondence to s.maddock@warwick.ac.uk
†Work was done while part-time at Meta AI

CCS '22, November 7–11, 2022, Los Angeles, CA, USA.

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560687

1 INTRODUCTION

It is well known that machine learning models can leak private information about individuals in the training set [14, 58]. Differential privacy (DP) [21] is a popular definition that has been developed to mitigate such privacy risks and has become the dominant notion of privacy in recent years. Much of the current research on private machine learning is focused on training deep learning models with differential privacy [1, 35, 41, 60]. DP is often combined with federated learning, where data resides on client devices, and only small information about model updates is collected from clients, in order to further minimize the privacy risk [36].

While deep learning models are powerful for a range of real-world tasks in a centralized setting, they are sometimes beaten by "simpler" models on tabular datasets. One such competitor is Gradient Boosted Decision Trees (GBDTs) [22, 31, 59] [32]. GBDT methods build an ensemble of weak decision trees that incrementally correct for past mistakes in training to improve predictions. Many GBDT frameworks such as XGBoost [16], LightGBM [38], and CatBoost [19] have seen widespread industry adoption [6, 11, 40, 44]. GBDT methods are an attractive alternative to deep learning due to their speed, scalability, ease of use, and impressive performance on tabular datasets.

Recent works have studied GBDT implementations such as XGBoost under secure training in the federated setting [17, 18, 24, 48]. These methods typically rely on cryptographic techniques such as Homomorphic Encryption (HE) or Secure Multi-Party Computation (MPC). While this allows secure joint training of a GBDT model without any participant directly releasing their data, the end model may not necessarily be private and will not guarantee formal differential privacy (DP) [23]. For instance, in the case of decision trees, split decisions in a tree can directly reveal sensitive information regarding the training set. Moreover, such reliance on heavyweight cryptographic techniques such as HE or MPC often makes methods computationally intensive or require a large number of communication rounds, making them impractical to scale beyond more than a few participants [17, 42].

In parallel, many works have studied decision tree models under the central model of DP [25, 52, 65]. Most studies focus on training random forest (RF) models and there has been little research to explore trade-offs between gradient boosting and DP; those that do often use central DP mechanisms that cannot easily be extended to federated settings [33]. It therefore remains an open problem to implement GBDTs in the federated DP setting, and show how to obtain utility comparable to their centralized non-private counterparts.

Our focus is on DP-GBDT methods that operate within the federated setting via lightweight MPC methods such as secure aggregation [7, 10]. This setting has recently risen to prominence, as it

promises an attractive trade-off between the computational efficiency of central DP techniques and the security of cryptographic methods. Recent federated works that consider GBDTs have proposed methods under the local model of DP, but due to the use of local noise, incur a significant loss in utility [43, 61, 62].

In this paper, we bring together existing methods under a unified framework where we propose techniques to satisfy DP that are well suited to the federated setting. We find that by dissecting the GBDT algorithm into its constituent parts and carefully considering the options for each component of the algorithm we can identify specific combinations that achieve the best balance of privacy and utility. We also emphasise variants that can train such private GBDT models in only a small number of communication rounds, which is of particular importance to the federated setting.

Our high-level finding is that it is possible to achieve high performance with GBDT models, even comparable to that of non-private methods. In order to do so, one must allocate privacy budget to the quantities that are most important for the learning process. For example, we show that spending such budget on computing split decisions of trees is not as important as spending it on the leaf weights. Using our findings under the efficient privacy accounting of Rényi Differential Privacy (RDP) leads to performance that is far closer to the non-private setting than seen in previous works.

Our main contributions are as follows:

- A clear and concise framework for differentially private gradient boosting with decision trees. We deconstruct the GBDT algorithm into five main components, showing how to federate each component while satisfying Rényi Differential Privacy (RDP). We present a unifying approach, capturing recently proposed DP tree-based models as special cases.
- A new set of techniques for improving the utility of private federated GBDT models. For example, we propose a private method for discretising continuous features that makes as much use of the private training information as possible, incurring little additional privacy cost. Additionally, we explore batching weight updates, showing it is possible to maintain competitive model performance while reducing the number of communication rounds needed.
- An extensive set of experiments on a range of benchmark datasets exploring the trade-offs between various options in our framework. By evaluating the choices in each of the components of our framework,we find a clear dominant approach is formed by adapting and simplifying the GBDT algorithm while combining it with our improved split candidate method. We show it is possible to achieve higher utility than state-of-the-art (SOTA) DP-RF and DP-GBDT methods on a range of datasets with reasonable levels of privacy.
- We provide open-source code at https://github.com/Samuel-Maddock/federated-boosted-dp-trees

Roadmap. In Section 2 we outline technical preliminaries required to understand differentially private GBDTs before covering related works in Section 3. In Section 4 and 5 we describe our framework for DP-GBDTs, fitting existing methods within this and proposing combinations to study. In Section 6 we provide extensive experimental evaluations, comparing our methods to existing baselines within our framework before concluding with Section 7.

2 PRELIMINARIES

2.1 Gradient Boosted Decision Trees (GBDT)

Tree-based ensemble methods form a collection of T decision trees that predict $\hat{y}_i$ for each input $\boldsymbol{x}_i$:

$$\hat{y}_i \coloneqq f(\boldsymbol{x}_i) = \sum_{t=1}^{T} f_t(\boldsymbol{x}_i)$$

For a specific tree f_t let L_t denote the number of leaf nodes. Each leaf node of a tree contains a weight, which will be the output of the tree for observations that are classified into that leaf. We denote $\boldsymbol{w}^{(t)} \in \mathbb{R}^{L_t}$ as the vector of leaf node weights for a tree f_t.

GBDT methods train trees sequentially making use of past predictions to correct for mistakes. This is in contrast to random forest (RF) methods that train T trees in parallel, averaging the weights of trees for the final prediction.

For a set of examples $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ with corresponding predictions $\{\hat{y}_i\}_{i=1}^{n}$ the GBDT objective function is defined as

$$\mathcal{L}(f) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t) \tag{1}$$

where ℓ is a twice-differentiable loss function, typically the cross-entropy loss (binary classification) or squared-error loss (regression). The term $\Omega(f_t) = \gamma L_t + \frac{\lambda}{2}||\boldsymbol{w}^{(t)}||_2^2$ is a form of regularisation such that $\gamma \geq 0$ penalises the size of the tree and $\lambda \geq 0$ penalises the magnitude of weights. This regularisation term is present in the popular XGBoost algorithm but is often omitted in other GBDT variants; we adopt it for our experimental study.

Equation (1) evades direct optimization. Rather, GBDT models are trained sequentially based on previous models. At step t we can define the model prediction $\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(\boldsymbol{x}_i) = \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)$. The objective for optimising f_t becomes

$$\mathcal{L}^{(t)}(f_t) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)) + \Omega(f_t) \tag{2}$$

For step t we are concerned with finding a tree f_t that minimises (2). Since f_t is not differentiable we can use a Taylor approximation. Taking the first-order approximation leads to the standard Gradient Boosting Machine (GBM) method. Taking a second-order approximation leads to Newton boosting as used by XGBoost [16].

When taking a first-order approximation we obtain

$$\mathcal{L}^{(t)}(f_t) \approx \sum_{i=1}^{n} \left(\ell(y_i, \hat{y}^{(t-1)}) + g_i^{(t)} f_t(\boldsymbol{x}_i) \right) + \Omega$$

where $g_i^{(t)} = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} \ell(y_i, \hat{y}_i^{(t-1)})$ is the gradient of the loss function at the start of step t. By considering the index sets of examples mapped to leaf node l i.e., $I_l = \{i | \boldsymbol{x}_i \text{ belongs to leaf } l \text{ of } f_t\}$ one can show by expanding the above and differentiating with respect to $w_l^{(t)}$ that the optimal leaf weight is

$$w_l^{(t)} = -\frac{\sum_{i \in I_l} g_i^{(t)}}{|I_l| + \lambda} \tag{3}$$

We denote this as a gradient weight update. Taking a second-order approximation of (2) instead gives

$$\mathcal{L}^{(t)}(f_t) \approx \sum_{i=1}^{n} \left(\ell(y_i, \hat{y}^{(t-1)}) + g_i^{(t)} f_t(\boldsymbol{x}_i) + h_i^{(t)} \frac{f_t^2(\boldsymbol{x}_i)}{2} \right) + \Omega(f_t)$$

where $h_i^{(t)} = \frac{\partial^2}{\partial (\hat{y}_i^{(t-1)})^2} \ell(y_i, \hat{y}_i^{(t-1)})$ is the Hessian of the loss at the start of step t. As before one can show that the optimal weights

this time are

$$w_l^{(t)} = -\frac{\sum_{i\in I_l} g_i^{(t)}}{\sum_{i\in I_l} h_i^{(t)}+\lambda} \tag{4}$$

which we denote as a Newton weight update. Substituting optimal weights from either the first or second-order approximation into Equation (2) leads to quantities that can be used to measure a split score. In other words, when considering a split option that partitions examples into disjoint index sets $I = I_1 \cup I_2$, the split score is a measure of how useful a split is for classification. The split score for Newton updates can be computed as

$$SS(I_1, I_2) = \frac{1}{2}\left[\frac{(\sum_{i\in I_1} g_i)^2}{\sum_{i\in I_1} h_i+\lambda} + \frac{(\sum_{i\in I_2} g_i)^2}{\sum_{i\in I_2} h_i+\lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i+\lambda}\right] - \gamma \tag{5}$$

In practice to form such split options, GBDT methods often discretize continuous features (e.g., via quantiles) into Q split candidates. In order to handle categorical features, GBDT methods like XGBoost typically transform them e.g., via a one-hot encoding. In either case, this leads to splits of the form $I_{\leq} = \{i : x_{ij} \leq s_q^j\}$ for a split candidate s_q^j. Equation (5) can then be used to greedily choose the feature split-candidate pair with the largest score when growing the tree structure during training.

2.2 Differential Privacy

Differential Privacy (DP) is a formal definition of privacy that guarantees the output of a data analysis does not depend significantly on a single individual's data item. Such a definition can be based on the notion of privacy loss.

Definition 2.1 (Privacy Loss Random Variable). Given a randomised mechanism $\mathcal{M} : \mathcal{X} \to \mathcal{Y}$ we define the privacy loss random variable $L_{\mathcal{M},x,x'}$ over "neighbouring" datasets $x, x' \in \mathcal{X}$ as

$$L_{\mathcal{M},x,x'} = \log\left(\frac{p_{\mathcal{M}(x)}(X)}{p_{\mathcal{M}(x')}(X)}\right)$$

where $X \sim \mathcal{M}(x)$ and $p_{\mathcal{M}(\cdot)}$ is the density of the mechanism applied to the respective dataset.

We take neighbouring datasets $x, x' \in \mathcal{X}$ to mean that x and x' differ on a single individual. The privacy loss allows us to succinctly describe differential privacy.

Definition 2.2 (Differential Privacy in terms of privacy loss [5]). We say that a randomised mechanism $\mathcal{M} : \mathcal{X} \to \mathcal{Y}$ satisfies (ϵ, δ)-DP if for any adjacent datasets $x, x' \in \mathcal{X}$

$$\mathbb{P}(L_{\mathcal{M},x,x'} \geq \epsilon) \leq \delta$$

The privacy parameter ϵ is referred to as the privacy budget. When $\delta = 0$, we say that $\mathcal{M}$ satisfies ϵ-DP. In this work we only consider privacy guarantees where $\delta > 0$ i.e., the case of approximate-DP.

While (ϵ, δ)-DP is a useful definition of privacy it does not allow us to tightly quantify the privacy loss from the composition of multiple mechanisms [37]. This is particularly important in machine learning where we wish to use mechanisms many times over the same dataset to train models. Instead, the notion of Rényi Differential Privacy (RDP) provides a succinct way to track the privacy loss from a composition of multiple mechanisms by representing privacy guarantees through moments of the privacy loss.

Definition 2.3 (Rényi Differential Privacy [49]). A mechanism $\mathcal{M} : \mathcal{X} \to \mathcal{Y}$ is said to satisfy (α, τ)-RDP if the following holds for any two adjacent datasets $x, x' \in \mathcal{X}$

$$\mathbb{E}\left[L_{\mathcal{M},x,x'}^{(\alpha-1)}\right] \leq \exp((\alpha-1)\tau)$$

One of the simplest and most widely-used mechanisms to guarantee (α, τ)-RDP is the Gaussian mechanism.

Fact 2.1 (Gaussian Mechanism [21, 49]). *The Gaussian mechanism $\mathcal{M} : \mathcal{X} \to \mathbb{R}^m$ of the form*

$$\mathcal{M}(x) = q(x) + N(0, \Delta_2(q)^2\sigma^2 I_m)$$

satisfies (α, τ)-RDP with $\tau = \frac{\alpha}{2\sigma^2}$ and

$$\Delta_2(q) = \max_{x,x'} \|q(x) - q(x')\|_2$$

The quantity $\Delta_2(q)$ is the L_2-sensitivity of the query q. The above shows that in order to make a real-valued query q differentially private we just need to add suitably calibrated Gaussian noise.

An attractive property of this formulation of DP is that it is easy to reason about the privacy of an analysis where mechanisms are used multiple times on the same dataset.

Fact 2.2 (Parallel Composition). *Given a dataset X, a disjoint partition $X = X_1 \cup X_2 \cdots \cup X_k$ and a mechanism $\mathcal{M}$ that satisfies (α, τ)-RDP. Then the mechanism $\mathcal{M}'(X) := (\mathcal{M}(X_1), \ldots, \mathcal{M}(X_k))$ satisfies (α, τ)-RDP.*

Fact 2.3 (Sequential Composition). *If $\mathcal{M}_1$ and $\mathcal{M}_2$ are (α, τ_1)-RDP and (α, τ_2)-RDP respectively then the mechanism that releases $(\mathcal{M}_1(\cdot), \mathcal{M}_2(\cdot))$ is $(\alpha, \tau_1 + \tau_2)$-RDP.*

Fact 2.4 (Post-Processing). *If $\mathcal{M}$ is an (α, τ)-RDP mechanism and f is any function that does not depend on any private data then $f(\mathcal{M}(\cdot))$ is also (α, τ)-RDP.*

Sequential composition tells us that using a mechanism multiple times on the same data leads to an increase in privacy loss. In the case of composing k Gaussian mechanisms, we must increase the noise added through σ by the order of $\sqrt{k}$ under RDP.

In practice, we care about obtaining the more meaningful notion of (ϵ, δ)-DP. When working with RDP we can rely on conversion lemmas such as those presented in [13] to convert between (α, τ)-RDP and (ϵ, δ)-DP. In our implementations, we use the analytical moment accountant developed by Wang et al. to provide tight numerical accounting of the privacy loss under RDP [63].

It is common to fix the privacy parameters (ϵ, δ) before the analysis and then minimises σ over a range of α values to obtain the smallest such noise needed to guarantee the chosen level of privacy. We use the autodp package[1] to verify our accounting provides the correct (ϵ, δ)-DP guarantee. An additional benefit of working with RDP is then that our framework easily extends to other mechanisms that satisfy RDP such as the Skellam mechanism which may be more suited to distributed settings [2].

[1] https://github.com/yuxiangw/autodp

Table 1: Summary of our Private Federated GBDT Framework

Component	Methods	Privacy Cost (in terms of $\kappa_s, \kappa_w, \kappa_c$)
(C1) Split Method	• Histogram-based (Hist) (§4.3.1)	$\kappa_s = Tmd$
	• Partially Random (PR) (§4.3.2)	$\kappa_s = Tmd$, does not require construction of a histogram
	• Totally Random (TR) (§4.3.2)	$\kappa_s = 0$
(C2) Weight Update	• Averaging (§4.4.1)	If using a Hist or PR $\kappa_w = 0$ otherwise $\kappa_w = T$
	• Gradient (§4.4.2)	
	• Newton (§4.4.3)	
(C3) Split Candidate	• Uniform, Log (§4.5.1)	Data-independent, $\kappa_c = 0$
	• Quantiles (non-private) (§4.5.1)	N/A
	• Iterative Hessian (IH) (§4.5.2)	If using Hist, $\kappa_c = 0$. If using TR, with s rounds of IH, $\kappa_c = sm$
(A1) Feature Interactions	• Cyclical k-way (§5.1)	If using Hist or PR, $\kappa_s = Tkd$, if $k = 1$ then $\kappa_s = T$.
	• Random k-way (§5.1)	If using TR with IH then $\kappa_c = sk$
(A2) Batched Updates	• $B = 1$ (Boosting) (§5.2)	Post-processing, no effect on privacy
	• $B = T$ (RF-type predictions) (§5.2)	
	• $B = p \cdot T$ for some $p \in (0, 1)$ (§5.2)	

2.3 The Federated Model of Computation

Federated Learning (FL) has become a popular paradigm for large-scale distributed training of machine learning models [36]. In this work, we consider the horizontal setting, where a set of participants each hold a local dataset over the same space of m features. We assume that there are n data items in total and we consider the problem of training a differentially private GBDT model over the distributed dataset. A powerful tool is secure aggregation, which allows the computation of a sum without revealing any intermediate values [7, 10]. Specifically, when each participant P_k has a number $x_k \in \mathbb{Z}$, secure aggregation computes the result $\sum_k x_k$ securely without any participant directly sharing their x_k.

Our focus is on a framework that combines secure aggregation with DP to securely and privately train GBDT models. For the rest of this paper, we present algorithms as if the data were held centrally, with the understanding that all the operations we use can be performed in the federated model (with rounding to fixed precision)[2]. This means that we avoid techniques designed for central evaluation such as the exponential mechanism [33, 45, 65].

Threat Model: In this work, in common with many other works in the federated setting, we assume an honest-but-curious model, where the clients do not trust others with their raw data. We study the aggregating server's knowledge based on the information gathered from clients. While there is potential for clients to attempt to disrupt the protocol, we leave the detailed study of more malicious threat models and model poisoning to future work. In order to combine secure aggregation with DP, we act as if there were a trusted central server that securely aggregates quantities and adds the required DP noise before sending the updated (private) model back to participants (as assumed in [47]). In practice, we can eliminate the need for a central server by well-established implementations of secure computation that rely on techniques from secure multi-party computation, either among a small number of honest-but-curious servers, or via clients working with small groups of neighbors and a single untrusted server [7]. Sufficient noise for DP guarantees can be added by honest-but-curious servers, or introduced by each client adding a small amount of discrete noise, such that the total noise across clients adds up to the desired volume [8, 9, 15, 53–57].

[2]The rounding introduces a small amount of imprecision in representing values, but this is overwhelmed by the noise added for privacy.

3 RELATED WORK

Differentially private decision trees have been well studied in the central setting with a strong focus on random forest (RF) models [25, 27, 65]. However, the boosted approach (i.e., private GBDT models) has been less well-explored. Recently, federated XGBoost models have been presented, with most works focused on secure training via cryptographic primitives such as Homomorphic Encryption (HE) and Secure Multi-Party Computation (MPC) and with no DP guarantees [17, 18, 24].

Some related works (e.g., [61]) study XGBoost in a federated setting with local DP (LDP) guarantees. The closest work to ours in this regard is the FEVERLESS method [62], which translates the XGBoost algorithm into the vertical federated setting using secure aggregation and the Gaussian mechanism. In particular, FEVERLESS securely aggregates gradient information into a private histogram which is used to compute split scores and leaf weights (Equations (4) and (5)). A certain subset of the participants are chosen as "noise leaders" to add Gaussian noise to their gradients information before aggregating to achieve an overall DP guarantee after securely aggregating across all participants. As we will see, the main disadvantage of directly translating the XGBoost algorithm in this way is the high privacy cost of repeatedly computing split scores. This results in having to add more noise into split score/leaf weight calculations and a lower utility model.

To reduce this privacy cost, one can consider making split decisions independently of the data. These so-called totally random (TR) trees have been studied in both the non-private and private settings with random forests [26, 29]. In the private setting, proposed methods often use central DP mechanisms that are hard to federate [3, 27]. For example, Fletcher and Islam [27] propose a DP-RF method that utilises the exponential mechanism to output the

majority label in leaf nodes under the notion of smooth sensitivity, which is unsuited to the federated setting.

In this work, we also consider TR trees as an option under our framework but for a federated and private GBDT model. To the best of our knowledge, the only other work that considers private boosting with random trees is that of Nori et al. [51]. They consider a central DP setting with a focus on training private explainable models via Explainable Boosting Machine (EBMs). We compare the technical differences in Section 5.1 and empirically in Section 6.6.

4 PRIVATE GBDT FRAMEWORK

In this section, we perform a comprehensive investigation of the main components needed to train GBDT models in the federated setting. We propose a framework of methods for training DP-GBDT models by identifying three main components that require DP noise and two additional components that interact with these. The full framework is summarized in Table 1.

We explain the various options in each component and how they affect privacy guarantees and conclude by instantiating related work into the framework before empirically evaluating methods in Section 6. A particular strategy we highlight is replacing data-dependent choices with random or uniform choices. Although counter-intuitive, it often holds that the privacy "cost" of fitting the choices to the data is not made up for by the utility gain, and picking among a set of random options is sufficient for good results. This is evaluated in our experimental study.

For simplicity, we assume that each participant holds a single data item $(\boldsymbol{x}_i, y_i)$ with n participants (data items) in total. We additionally assume that we have (publicly) known bounds on each feature. All of these assumptions can be easily removed, potentially with some additional privacy cost. Table 7 in the appendix displays commonly used notation for convenience.

4.1 A General Recipe

In order to train the GBDT algorithm outlined in Section 2.1 we only need to specify a few core choices: How to pick split candidates (for discretizing continuous features), calculate the split scores at each internal node, and compute the leaf weights for prediction. One can note from Equations (4) and (5) that the leaf weights and split scores only depend on the sum of gradients and Hessians at an internal or leaf node of a decision tree. It is therefore natural to utilise secure aggregation as a tool to federate the GBDT algorithm. In Algorithm 1 we present the general GBDT algorithm assuming these quantities can be gathered. Looking closely at Algorithm 1, the only time we need to directly query participants' data is when we compute the three quantities just mentioned.

Based on this we divide the general algorithm into 3 core components that require some form of DP noise: Split Methods **(C1)**, Weight Updates **(C2)**, and Split Candidates **(C3)**. These are the core components required for training a GBDT model. We also consider two additional aspects to specify when training a GBDT model: Feature Interactions **(A1)** and Batched Updates **(A2)**. These are aspects that interact with the core components but do not require any additional noise. To reason about the privacy guarantees of our GBDT framework, we introduce some variables to count the number of queries needed when training a GBDT model with T trees. Let κ_c denote the number of queries needed to calculate split candidates; κ_s for the queries needed to calculate inner node splits; and κ_w for the queries to calculate leaf weights. Counting the number of queries needed for each component is enough to give a privacy guarantee for Algorithm 1.

Algorithm 1 General GBDT

Input: Number of trees T, maximum depth d, number of split candidates Q, privacy parameters ϵ, δ

1: For each feature $j = 1, \ldots, m$ generate Q split candidates $S_j := \{s_1^j, \ldots, s_Q^j\}$ **(C3)**
2: Initialise the forest $\mathcal{T} \leftarrow \emptyset$
3: **for** $t = 1, \ldots, T$ **do**
4: For each $(x_i, y_i) \in D$ compute the required gradient information (g_i, h_i) based on $\hat{y}_i^{(t-1)}$ **(C2)**
5: Choose a subset of features $F^{(t)} \subseteq \{1, \ldots, m\}$ with $|F^{(t)}| = k$ for the current tree f_t **(A1)**
6: **while** depth of the current node (in f_t) is $\leq d$ **do**
7: Choose a feature split candidate pair (j, s_q^j) from $F^{(t)}$ **(C1)**
8: Split the current node with observations I into two child nodes with index sets $I_{\leq} = \{i : x_{ij} \leq s_q^j\}$ and $I_{>} = I \setminus I_{\leq}$
9: Repeat (6)-(9) recursing separately on the child nodes
10: For each leaf l calculate a weight $w_l^{(t)}$ from the examples in the leaf according to the chosen update method **(C2)**
11: Update predictions $\hat{y}_i^{(t)}$ or batch updates **(A2)**
12: Add the tth tree f_t to the ensemble, $\mathcal{T} = \mathcal{T} \cup \{f_t\}$
13: **return** the trained forest $\mathcal{T}$

Theorem 4.1. *Suppose that each mechanism for the framework components satisfies $(\alpha, \tau_c), (\alpha, \tau_s), (\alpha, \tau_w)$-RDP respectively. Then the GBDT algorithm satisfies (α, τ)-RDP with $\tau = \kappa_c\tau_c + \kappa_s\tau_s + \kappa_w\tau_w$.*

The above simply follows from the sequential composition properties of RDP. In our experimental study we utilise the Gaussian mechanism for each core component, hence $\tau = (\kappa_c + \kappa_s + \kappa_w)\frac{\alpha}{2\sigma^2}$ and so $\sigma = O_\epsilon(\sqrt{\kappa_c + \kappa_s + \kappa_w})$. This shows that if we can minimise the number of queries that each main component requires, then we reduce the amount of noise we add to the learning process while still maintaining privacy. Various methods affect the privacy cost in different ways. The privacy implications for different choices in terms of $\kappa_c, \kappa_s, \kappa_w$ are shown in Table 1.

4.2 Federating GBDTs

At the start of building the t-th tree, each participant calculates gradient information $(g_i^{(t)}, h_i^{(t)})$ for their examples. Throughout the training of a single tree, to calculate the desired components we can rely on querying data in the form $q(I) = \left(\sum_{i \in I} g_i^{(t)}, \sum_{i \in I} h_i^{(t)}\right)$ over some set of observations I, e.g., all observations in a specific tree node. To do this securely, we can apply secure aggregation to

aggregate gradient information at the various stages that require it in Algorithm 1 (**C1- C3**).

In order to apply the Gaussian mechanism, we must bound the sensitivity of such a query function. In this case, we need bounds on the gradient quantities $g_i^{(t)}, h_i^{(t)}$. Our focus in this paper is on binary-classification problems. In binary-classification our loss function is of the form $\ell(y_i, \hat{y}_i) = y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$ (i.e., binary cross-entropy) and has gradients $g_i \in [-1, 1]$ and Hessians $h_i \in [0, \frac{1}{4}]$. Hence the sensitivity of aggregating gradient information is $\Delta_2(q) = \sqrt{1 + \frac{1}{16}} = \frac{\sqrt{17}}{4}$.If the chosen loss function has unbounded gradient information (e.g., regression problems) we can employ gradient clipping (similarly to DP-SGD) to obtain a bounded sensitivity [1].

The computational and communication costs of these steps are low. Decision tree-based methods are often preferred for their ease of construction, and this translates to the federated setting: each client computes its local updates (e.g., gradients and Hessians) and shares these through secure aggregation. The communication costs are linear in the size of the updates computed, which are fairly low dimensional: we quantify this in the subsequent sections.

4.3 Component 1: Split Methods

4.3.1 Greedy Approach: Histogram-Based. As described in Section 2.1, the standard GBDT algorithm will calculate Q split-scores for every feature j. This forms feature-split pairs (j, s_q^j) and at each internal node the pair with the highest score is chosen to grow the tree. This split score depends on aggregating gradient and Hessian values. The most suitable way to do this in a federated setting is to form a histogram over the split candidates for every feature. This requires (securely) aggregating the gradient and Hessian values into bins partitioned by the split candidate values. Hence the q-th gradient histogram bin for feature j contains $G_q^j = \sum_{i \in \{i : s_{q-1}^j < x_{ij} \leq s_q^j\}} g_i^{(t)}$, and similarly for Hessians.

We can apply our generic aggregation query $q(I)$ with $I = \{i : s_{q-1}^j < x_{ij} \leq s_q^j\}$ to aggregate bins of both the gradient and Hessian histograms. Each participant's data item will fall into exactly one histogram bin, so via parallel composition we just need to count the number of times a histogram is computed during training. At each internal node of a tree, we must compute split-scores and thus gradient histograms. When considering all m features per split, this requires $\kappa_s = Tmd$ queries for a model with T trees of maximum depth d. This incurs a high privacy cost for large ensembles[3]. Each client can quickly compute and send their histograms of size Q for each feature considered for a split.

4.3.2 Randomised Approach: Partially and Totally Random. In [29], Geurts et al. initiate the study of "Extremely Randomised Trees" (ERTs) in the non-private setting. In ERTs the idea is to add randomness into the split choices when growing the tree. The motivation was to show that accuracy comparable to that of greedy tree-building models could be obtained for large enough ensembles. ERTs are potentially much faster to train as there is no need to compute split scores for each internal node. This leads to two pragmatic choices for splitting nodes:

- **Partially Random (PR)**: For each feature j pick a split candidate $s_q^j \in S_j$ uniformly at random, where S_j is the set of split candidates for j. The split score of (j, s_q^j) is computed for each feature and the pair with the highest score is chosen. This still requires $\kappa_s = Tmd$ queries but does not require building histograms.
- **Totally Random (TR)**: Pick a feature $j \in [m]$ and a split candidate s_q^j, both uniformly at random. This does not require any queries for internal nodes ($\kappa_s = 0$) as it is data independent.

Since TR trees do not access data to build tree structure they are attractive from a privacy perspective. All trees in the ensemble can be pre-computed by choosing random splits, which can be communicated to clients at a cost linear in the size of the tree. Hence building a TR ensemble requires far fewer queries than histogram-based methods. However, a TR ensemble often requires a much larger number of trees to achieve similar model performance as histogram-based counterparts. We explore such trade-offs between TR and histogram-based methods in Section 6.2.

4.4 Component 2: Weight Updates

Once a tree has been built, the records in the dataset will be partitioned among the leaf nodes of the tree. In the following we consider the l-th leaf of tree t with weight $w_l^{(t)}$ which contains records $I_l^{(t)} = \{i \in [n] : i \text{ belongs to leaf } l \text{ of tree } t\}$. Each client needs to compute and send the weights of leaf nodes, at cost proportional to the number of leaves, 2^d if there are d binary splits. Both RF and GBDT methods update these leaf nodes with a weight that contributes to prediction. As we noted in Section 2.1, taking a first-order or second-order approximation to Equation (2) leads to two different weight updates. Note that by setting $h_i = 1$ in both (4) and (5) we recover the gradient weight update of Equation (3) and also obtain a split score for gradient updates. Hence when $h_i = 1$ both approaches are equivalent and so Newton updates can be seen as generalising the standard gradient approach. While RFs do not calculate gradient information, we can still view them as a special case within our framework. RF trees typically compute the class probabilities in leaf nodes which are averaged across all trees in the ensemble. This leads to three main weight updates: zeroth-order (Averaging), first-order (Gradient), and second-order (Newton).

4.4.1 Averaging Updates. For random forests the leaf nodes store the class distribution. For regression problems this is the average value of y in the leaf node. With binary classification, the weight update is simply the proportion of positive examples in the leaf node i.e., $w_l^{(t)} = \frac{1}{|I_l^{(t)}|} \sum_{i \in I_l^{(t)}} \mathbb{1}\{y_i = 1\}$.

Although RF models do not compute gradients we can still utilise our generic aggregation query by having participants send $g_i = \mathbb{1}\{y_i = 1\}$ and $h_i = 1$. In this case $\sum_{i \in I} g_i$ counts the number of class 1 examples and $\sum_{i \in I} h_i$ counts the number of examples in a node. This changes the sensitivity of our query to $\Delta(q) = \sqrt{2}$.

In RF models the trees are independent from one another with final predictions formed from the average of weights across all trees. We denote this as an averaging update from now on.

[3]Default XGBoost parameters take $d = 6$ and $T = 100$ which implies a high privacy cost on any dataset with a moderate number of features m

4.4.2 Gradient Updates. Each participant calculates $g_i = \frac{\partial}{\partial \hat{y}_i}\ell(y_i, \hat{y}_i)$ and $h_i = 1$ and uses this in the weight update defined in Equation (3), i.e., the weights are the average negative gradient values in the leaf node. This can be viewed as a gradient descent step over the batch of observations in leaf node j. The sensitivity of the query also changes to $\Delta(q) = \sqrt{2}$.

4.4.3 Newton Updates. Participants calculate both first-order and second-order gradients of the form $g_i = \frac{\partial \ell}{\partial \hat{y}_i}, h_i = \frac{\partial^2 \ell}{\partial (\hat{y}_i)^2}$ and use the weight update in Equation (4).

In classification problems the total weight across trees for an observation i is aggregated and the sigmoid function $\sigma(\cdot)$ is applied. It is also standard in GBDT methods to perform post-processing on leaf weights. In practice we consider updates of the form $-\eta \cdot \max\{w_l^{(t)}, \beta \cdot \text{sgn}(w_l^{(t)})\}$ where $\eta > 0$ is the learning rate and $\beta \geq 0$ is a clipping factor to control the magnitude of updates.

For histogram-based splitting, the final gradient histograms from the parent of a leaf node can be used to calculate weights, meaning $\kappa_w = 0$. For TR splitting, participants do not calculate histograms so they must directly aggregate the required gradient information in each leaf node. This is a single query per leaf node that happens once per tree, and so $\kappa_w = T$.

4.5 Component 3: Generating Split Candidates

One major step needed to train GBDT models is to identify split candidates for each (continuous) feature. In traditional GBDT models such as XGBoost, split candidates are chosen by computing the quantiles of a feature. Computing quantiles is a succinct way to describe a feature's distribution but can be slow in practice for large datasets. The original XGBoost paper proposes a weighted quantile sketch to make this process faster, using the Hessian information as weights. While this is suitable in non-private settings, it is difficult to calculate such quantiles (or quantile sketches) accurately without incurring an appreciable privacy cost. Existing work on DP-GBDTs has computed split candidates either with LDP quantiles in the local setting [43], DP quantiles in the central setting [33] or with MPC methods (without DP guarantees) in distributed settings [61].

4.5.1 Data-Independent Split Candidates. The simplest and cheapest (from a privacy perspective) approach is to propose split candidates independently of the data, such as via uniform splits. For a feature j with values in $[a, b]$, one can generate a split candidate for each $q \in [Q]$ uniformly over $[a, b]$ as $s_q^j = a + (q-1)(b-a)/(Q-1)$. As we assume bounds on features are public knowledge, we do not need to query participants' data, and hence $\kappa_c = 0$.

A disadvantage of this approach arises when features are heavily skewed as uniform splits are unlikely to cover important areas of the feature's distribution. One possible approach would be to take a log transform of skewed features and then split uniformly over the transformed feature. In the non-private setting, one can manually check features or use statistical skewness tests to determine when to transform features. This poses a problem in the private setting as we may not know a priori which features are skewed and privately computing such a test may be expensive privacy-wise.

4.5.2 Iterative Hessian (IH) Splitting. We propose an alternative method based on making use of information that is usually calculated during the training process. We will verify for datasets with heavily skewed features that we can achieve similar AUC to optimal non-private split candidate methods for little to no additional privacy cost. Specifically, the Hessian information in Newton boosting captures the certainty of predictions and is often used in the non-private setting to guide quantile finding. We can take a similar approach in the private setting provided we estimate aggregated Hessian values at each split candidate bin i.e., a Hessian histogram. We propose the following intuition to propose new split candidates at each round:

- **Merge bins** with low (or zero) Hessian since this indicates a split is too fine-grained to be useful.
- **Split bins** that have large Hessian value as this indicates a large number of observations lie in the bin. To refine a bin we can split by taking the midpoint of adjacent bin edges.

In practice, we split a bin if its Hessian value is greater than the total Hessian uniformly divided over the Q bins. If at the end of a round we end up with fewer than Q bins, then we fill the remaining bins by uniformly splitting. The full algorithm for IH is given in Algorithm 2 in the appendix. Carrying out IH splitting is a form of post-processing on the Hessian histogram and thus has no extra privacy cost beyond the cost to compute the histogram. However, the choice of split method may incur additional privacy cost:

- *Hist:* In histogram-based methods, a Hessian histogram is computed at the start of every tree for all features. We can use the previous tree's Hessian information to inform our split candidates for each new tree. We incur no additional privacy cost and hence $\kappa_c = 0$
- *Totally random:* As TR trees are built independently of the data, Hessian histograms are never computed. We propose to calculate a Hessian histogram for the first s rounds of training and thus the number of queries we need for split candidates is $\kappa_c = sm$. For the first s rounds we refine our split candidates using IH, after which we use the final set of candidates found in round s for the remaining $T - s$ trees.

5 ADDITIONAL CONSIDERATIONS

5.1 Feature Interactions

Explainable Boosting Machines (EBMs) are a popular method for training GBDTs to ensure explainability of the resulting model [46]. The main idea is to construct an additive model of the form $f(\boldsymbol{x}) = \sum_{j=1}^{m} \alpha_j f_j(x)$ where each $f_j(x)$ is a boosted decision forest with trees that are trained only on the j-th feature. Nori et al. [51] consider the problem of training DP-EBM models in the central setting. Their method relies on training many very shallow trees with totally-random (TR) splits. In order to ensure explainability, each tree of the ensemble is restricted to a single feature at a time. This results in a "cyclical" boosting method where tree t is trained only on feature $j = t \mod m$. Although the focus of our work is not on explainability, Nori et al. note that the cyclical training method of EBMs actually results in more accurate models (with DP) when compared with models that can freely split on all features per tree.

This presents another design decision—whether to train trees cyclically (so that each tree only splits on a single feature at a time), or to train trees that consider a subset of k features to choose from when splitting a node. We define k-way feature splitting as considering k features at a time per tree. This can be done in two different ways:

1. Cyclical k-way: Consecutive trees train on a subset of the next k features and repeat in cycles every m/k trees.

2. Random k-way: The k features are chosen randomly at the start of each tree.

When $k = 1$ with cyclical training we recover the method used in EBM. When $k = m$ we recover the standard GBDT splitting method. The choice of k determines the maximum number of feature interactions that are possible within a tree. We note that the random k-way method is also commonly used in the non-private setting to reduce computation time and act as model regularisation [28]. The computation and communication costs for each client scale proportionally to k. For histogram-based methods with $k > 1$ the number of queries required to form internal node splits for k-way splitting is $\kappa_s = Tkd$. When $k = 1$ this reduces to $\kappa_s = T$ and $\kappa_w = 0$ since gradient histograms can be computed once at the root node and this same histogram can be used to calculate split-scores for every level in the tree. For totally-random trees the value of k does not affect the number of queries and $\kappa_s = 0$ remains. In Appendix A.3 we present experiments detailing the effect of feature interactions. We observe that cyclical $k = 1$ (i.e, EBM) performs the best and we focus on this in our end-to-end comparison in Section 6.6.

5.2 Batched Updates

One advantage of random forests (RF) in distributed settings is that trees can be trained in parallel. In the case of totally random (TR) trees the model orchestrator can precompute the structure of all trees and participants can compute gradient statistics for leaf weights over the entire forest in a single round of communication. On the other hand, gradient boosting methods are inherently sequential—results of the previous ensemble determine the gradient calculations for the next tree. This is a bottleneck for weight updates. One way to parallelise this is to consider batching updates.

Suppose we use a batch size of B and are training T trees. A batched update is of the form

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-B)} + \eta\sigma\left(\frac{1}{B}\sum_{k=t-B}^{B} w_i^{(k)}\right)$$

where we abuse notation to let $w_i^{(k)}$ denote the weight of the leaf in tree k that $\boldsymbol{x}_i$ is partitioned into.

Batched updates require participants update their predictions every B rounds based on the average leaf weight of the trees in the batch. At the start of the $(B+1)$-th round the gradient information is recomputed so that the next batch is boosting predictions from the previous batch. One can think of this as boosting a set of B-sized random forests. When $B = T$ we recover RF-type predictions but note that if $w_l^{(t)}$ uses gradient or Newton weights then the model updates are different from averaging updates (which instead use class probabilities as weights). Batching updates also has no extra privacy cost as it is a form of post-processing.

If we wish to train T trees with a batch size B then the communication rounds of the boosting process reduce from $O(T)$ to $O(T/B)$[4]. In Section 6.5 we consider batching gradient and Newton updates for different batch sizes and compare to DP-RF which requires $O(1)$ rounds of communication [26].

5.3 Instantiating the GBDT Framework

5.3.1 Instantiating Components. In the previous sections, we have deconstructed the GBDT algorithm into various core components that require us to add noise to guarantee DP. We also noted two additional considerations that interact with the core components. When instantiating our framework in experiments we will combine the options for each component as listed in Table 1. A key contribution of this work is in the comprehensive study of both existing (Section 5.3.2) and new (Section 5.3.3) methods as follows:

(C1) Split Methods: The histogram over split candidates is how centralized algorithms like XGBoost structure the problem [16]. It is also friendly to federation and has been used by prior works so forms a natural baseline [17, 24, 62]. Totally random trees have been widely used in non-private RF models but have not been well-studied in private, federated and GBDT settings [29]. DP-EBM is the only prior example we are aware of here [51]

(C2) Weight Updates: We consider standard update methods used in GBDTs/RFs noting that Hessian updates have not been as well-studied under privacy or the federated setting.

(C3) Split Candidates: Data-independent splits have been largely overlooked in central DP settings with effort put into calculating DP quantiles. We advocate it as a good option for the federated setting since the (privacy) cost of finding quantile splits is not repaid in practice. We introduced the Iterative Hessian (IH) approach based on refining candidates over a number of rounds which helps when features are particularly skewed.

(A1) Feature Interactions: The idea of (maximum) feature interactions generalizes the Explainable Boosting Machines (EBM) method which considers a single feature per tree [51].

(A2) Batched Updates: The idea of batching updates has not been studied in the private and federated setting. It can be viewed as boosting individual RFs which is sometimes done in non-private settings. Our focus here is on reducing communication rounds while still maintaining accuracy.

5.3.2 Instantiating Related Work. In Table 2 we outline how SOTA DP-GBDT models can be expressed in our framework. These act as the primary baselines in our experiments. We note that many of these methods were originally proposed to use pure ϵ-DP in the central setting and often rely on basic composition results. We have re-implemented all methods to use tighter RDP accounting and guarantee (ϵ, δ)-DP so they are not disadvantaged. To summarise:

- **DP-EBM [51]** is a DP variant of the EBM model. It uses Gaussian Differential Privacy (GDP) but as this is known to under-report ϵ values [30], we use RDP in our experiments. DP-EBM uses TR splits with gradient updates, where each tree only considers a single feature. The split candidate method is a central DP histogram that attempts to uniformly distribute observations among bins. We replace this with uniform split candidates in our experiments.
- **DP-RF [26]** is a central DP method that builds a RF via TR splits. The method was originally proposed for categorical features and

[4]Ignoring the constant number of rounds required for secure aggregation. See Appendix B for more details

Table 2: Related works under our framework

	DP-EBM [51]	FEVERLESS [62]	DP-RF [25]
C1: Split Method	TR	Hist	TR
C2: Weight Update	Gradient	Newton	Averaging
C3: Split Candidate	Uniform (DP Hist)	Quantile Sketch	N/A
A1: Feature Interactions	Cyclical ($k = 1$)	m-way	m-way
A2: Batched Updates	$B = 1$	$B = 1$	$B = T$
$\kappa_c + \kappa_s + \kappa_w$	$0 + 0 + Tm$	$0 + Tmd + 0$	$0 + 0 + T$

later extended to continuous features [27]. The Laplace mechanism is used to perturb leaf weights we re-implement this using the Gaussian mechanism under RDP accounting. In our federated framework, DP-RF corresponds to using TR splits, the averaging weight update, and uniform split candidates (with $k = m$, $B = T$).

- **FEVERLESS [62]** corresponds to a Hist split method with Newton weight updates. FEVERLESS uses a quantile sketch which is non-private in our horizontally partitioned setting; we replace this with uniform splits to make FEVERLESS fully private.

In our final comparisons in Section 6.6 we compare to an LDP baseline. This baseline has each user add Gaussian noise before releasing their gradient information. Such noise only needs to be scaled by the number of trees (T) since each user can release noised gradients at the root node, and the server can use this to construct the tree. While LDP does not strictly fall into our framework, it is a useful benchmark to compare against distributed DP counterparts.

5.3.3 Instantiating Other Methods. We end Section 6 with an end-to-end comparison of the baselines against new combinations expressed under our framework. These methods are:

- **DP-EBM Newton,** the DP-EBM method with Newton updates instead of Gradient updates. We also do not train Tm trees but only T. The total privacy cost here is $\kappa_c + \kappa_s + \kappa_w = 0 + 0 + T$
- **DP-TR Newton,** the TR spit method, uniform split candidate and Newton updates. The privacy cost is the same as DP-EBM.
- **DP-TR Newton IH EBM,** a DP-TR Newton with EBM feature interactions (i.e., cyclical $k = 1$). The privacy cost is $\kappa_c + \kappa_s + \kappa_w = 0 + 0 + T + sm$ where s is the number of rounds IH is performed.
- **DP-TR Batch Newton IH EBM** ($p = 0.25, p = 1$)**,** i.e., DP-TR Newton IH EBM with batched updates with $p = 0.25$ or $p = 1$, the privacy cost is the same as DP-TR Newton IH EBM.

6 EMPIRICAL EVALUATION

Sections 4 and 5 introduced our framework for the private and federated training of GBDT models. In this section we perform a thorough experimental evaluation of the components in our framework. Our main goal is to answer the following questions:
1. In terms of model performance, what are the best options for each component under our framework?
2. Under privacy, does batching updates improve performance?
3. Can a combination of choices in our framework result in methods that improve over the SOTA baselines discussed in Section 5.3.2?

Figure 1 shows a snapshot of our findings. We display for a subset of datasets and methods, the average test AUC while fixing the privacy budget $\epsilon = 0.5$. Full results across all datasets are discussed in Section 6.6. We represent baseline methods plotted as circles and

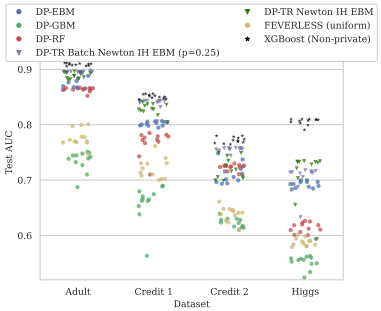


Figure 1: Snapshot of results for representative methods studied in Sec 6.6. $T \in [25, 300], \epsilon = 0.5, d = 4$.

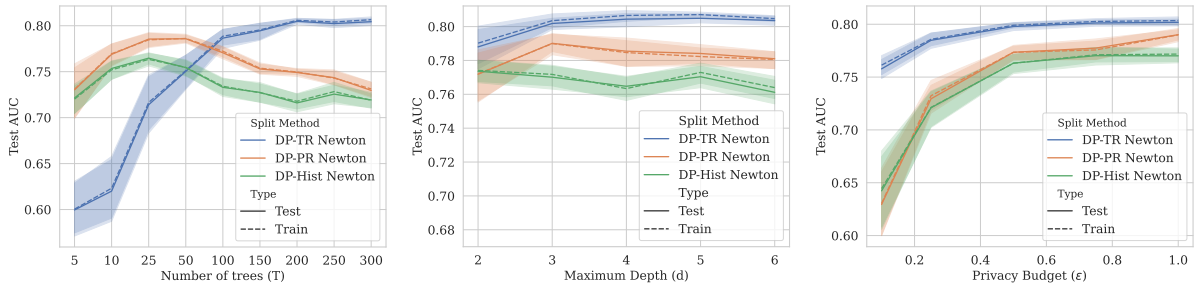
new combinations within our framework as triangles. Each point is formed from varying $T \in [25, 300]$ in increments of 25 and is the average test AUC[5] over 5 runs. We observe that on most datasets we significantly improve over existing methods. In some cases we match the nearest competitor, but often with additional benefits such as reducing the number of rounds of communication.

These experimental results, along with others in this section, show that it is possible to train accurate, private and lightweight federated GBDT models with only a small gap behind their non-private counterparts. This conclusion is reached by answering our questions as follows:
1. In Sections 6.2—6.4 we evaluate the choices within each component. We find that the totally-random strategy provides a significant reduction in privacy cost and outperforms all other choices. For weight updates we find that utilising Hessian information usually gives better performance with no additional cost, which is similar to the non-private setting. Finally, for split candidates, we find our IH method achieves performance that matches that of (non-private) quantiles with little extra privacy cost.
2. In Section 6.5 we study batching updates to help reduce the number of communication rounds. We find this is not the only benefit of batching and in fact, for very high-privacy regimes ($\epsilon < 0.5$), batching updates often gives better model performance than performing boosting for the full T rounds.
3. In Section 6.6 we combine the best individual components and compare against our SOTA baselines. We find combining the best options found in each component also results in the best model overall. Specifically, combining batched updates, the IH split candidate method, TR splits and Newton updates often achieves better performance than the most competitive baseline (DP-EBM) and in fewer rounds of communication.

6.1 Experimental Setup

In our experiments, we use a range of real-world datasets from Kaggle [34, 64] and the UCI repository [20]. All datasets are displayed in Table 8 in the appendix detailing the number of records (n), number of features (m), and proportion of the positive class (p). The Higgs dataset has been subsampled to $n = 200,000$ for computational reasons. All experiments are repeated 3 times over

[5] Due to class imbalance, measures such as accuracy are not useful to test performance.



(a) Varying T with $d = 4, \epsilon = 1$ **(b) Varying d** **(c) Varying ϵ**

Figure 2: Split Methods on Credit 1

Table 3: Weight update methods across the datasets fixing $\epsilon = 0.5, d = 4$

		Bank	Credit 1	Credit 2	Adult	Nomao
Hist ($T = 25$)	Gradient	0.6282 +- 0.0688	0.5748 +- 0.0852	0.6288 +- 0.0569	0.6749 +- 0.0524	0.8483 +- 0.0138
	Averaging	0.7249 +- 0.0274	0.6769 +- 0.058	**0.6751 +- 0.0246**	0.6373 +- 0.0457	**0.8885 +- 0.0038**
	Newton	**0.7562 +- 0.0337**	**0.7522 +- 0.0162**	0.6575 +- 0.0486	**0.8013 +- 0.0225**	0.8758 +- 0.0075
PR ($T = 25$)	Gradient	0.676 +- 0.0376	0.7094 +- 0.0312	0.6239 +- 0.0486	0.7688 +- 0.0253	0.8766 +- 0.0072
	Averaging	0.7803 +- 0.0309	0.7165 +- 0.0337	0.6864 +- 0.0249	0.8281 +- 0.0183	**0.8904 +- 0.0055**
	Newton	**0.7998 +- 0.0203**	**0.7676 +- 0.0196**	**0.6882 +- 0.0207**	**0.8416 +- 0.0108**	0.88 +- 0.0072
TR ($T = 300$)	Gradient	**0.8508 +- 0.0061**	0.7847 +- 0.0097	**0.7392 +- 0.008**	**0.8737 +- 0.0056**	**0.8965 +- 0.0047**
	Averaging	0.8382 +- 0.0116	0.7846 +- 0.0106	0.7285 +- 0.0109	0.8666 +- 0.0043	0.8875 +- 0.0055
	Newton	0.8486 +- 0.0075	**0.7983 +- 0.0062**	0.7344 +- 0.0088	0.8718 +- 0.0049	0.8883 +- 0.007

5 different 70-30 train-test splits resulting in 15 iterations. We measure model performance by the AUC-ROC on the test-set. For all boosting experiments we fix the learning rate and regularization parameters $\beta = 2, \eta = 0.3, \alpha = 0$ which generally performed well across all chosen datasets, and do not tune these any further. We take $Q = 32$ split candidates unless otherwise stated. The effect of the number of split candidates is explored in Section 6.4. In all experiments we use RDP to satisfy (ϵ, δ)-DP fixing $\delta = 1/n$. Tests were run with an AMD Ryzen 5 3600 3.6GHz CPU and 16GB of RAM. Code for our framework and experiments is open-sourced [6]

6.2 Split Methods

We begin by exploring the initial trade-off between the main split-methods: Histogram-Based (Hist), Partially Random (PR), and Totally Random (TR). We study these split methods as we vary parameters that have the largest effect on the AUC of DP-GBDT algorithms: T, d, and ϵ. For now we fix our weight update method to Newton and fix the split candidate method to uniform. We consider the effects of these components separately in Sections 6.3 and 6.4.

Figure 2a shows the effect of varying the number of trees T while fixing $\epsilon = 1, d = 4$ on the Credit 1 dataset, and visualises the key differences between the main split methods. Other datasets using the same parameter setup are considered in Appendix A.1. Recall that histogram-based and PR are methods that compute split-scores under DP. Because they compute split-scores they often "converge" to their best test AUC before TR methods in the non-private setting. We can observe that a similar effect occurs in the private setting. We see that PR and Hist peak around $T = 25 - 50$ whereas it takes TR $T = 300$ trees to achieve its best test AUC.

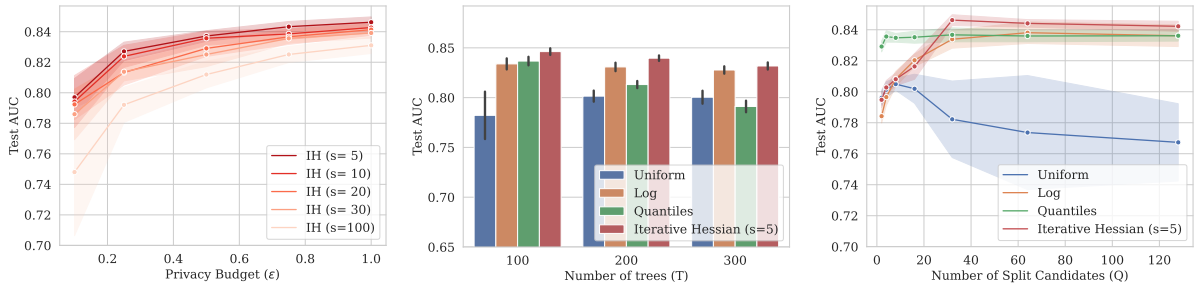
In the non-private setting this peak is typically caused by overfitting as T gets larger. For the private setting this is not quite the case as we can observe little difference in train and test AUC. Instead, for large T the privacy cost of training a histogram-based GBDT model requires a large amount of noise to be added at each step and this severely impacts performance.

Recall that both Hist and PR split methods require Tdm queries to train the full model compared to just T for TR. The advantage of TR's minimal privacy cost can be clearly seen from Figure 2a as it achieves higher AUC than the other two methods.

In Figure 2b we fix $\epsilon = 1$ and set $T = 25$ for Hist/PR and $T = 300$ for TR as we vary the maximum depth $d \in \{2, 3, 4, 5, 6\}$ on Credit 1. We observe only a small difference in AUC across Hist method and only a minor decrease in performance across TR and PR methods for larger depths. For PR and Hist the depth d does increase the privacy cost of each tree but for TR the privacy cost is independent of the depth. We observe a small decrease in AUC for TR as d increases and this is likely because training very deep trees can lead to nodes with only a few observations. This results in gradient information with magnitude smaller than the noise being added, and hence any meaningful information is lost.

In Figure 2c we vary $\epsilon \in \{0.1, 0.25, 0.5, 0.75, 1\}$ while fixing $d = 3$. We set $T = 300$ for TR and $T = 25$ for Hist and PR. We can immediately make two observations. First, there is still a clear gap

[6] https://github.com/Samuel-Maddock/federated-boosted-dp-trees



(a) Varying s with $T = 100, d = 4$ **(b) Varying T with $d = 4, \epsilon = 1$** **(c) Varying Q with $T = 100, d = 4, \epsilon = 1$**

Figure 3: Split Candidate Methods on Credit 1

in performance between TR and Hist/PR. Second, for large ϵ, PR outperforms the Hist method but for small ϵ the picture is less clear. This is likely due to the additional random variation due to the PR method picking random split candidates.

Summary. We recommend using TR splits as it clearly outperforms methods that calculate split scores. This usually results in larger ensembles which can be prohibitive in federated settings. In Section 6.5, we discuss how we can batch updates to get around this.

6.3 Weight Update Methods

We start this section by asking whether boosted decision trees under DP provide any additional model performance over DP-RFs. Table 3 shows the test AUC across all datasets varying the weight-update method (Gradient, Averaging and Newton) for each split method. In these experiments we fix $\epsilon = 1, d = 4$ and use $T = 25$ for PR/Hist and $T = 300$ for TR methods. The highest AUC for each split method is highlighted in bold.

We can observe that boosting does provide an advantage over the traditional averaging method on these datasets, although it is not completely clear cut. Focusing first on the Hist methods we can see that Newton updates perform the best across three of the five datasets – although results on Credit 2 and Nomao show averaging performs the best. However, Newton updates certainly show clearer advantages on Credit 1, Adult, and Bank over both gradient and averaging updates. This pattern is also present for PR methods with Newton updates performing better than averaging except for Nomao where averaging performs the best. For TR methods we observe Gradient updates achieve higher AUC on 4 out of 5 of the datasets, although is within random variation of Newton for all datasets except Credit 1, where Newton performs best. We also note that the gap in performance between TR and Hist/PR observed in Section 6.2 also holds across all the datasets we are considering. The impact in performance between Newton and the other weight update methods for TR splits is also less marked than its impact with Hist/PR splits, since the performance of TR with Newton differs by at most 0.014 AUC when compared with gradient or averaging.

Summary. We recommend using Newton updates as it exceeds or performs very similarly to Gradient updates and in most cases beats averaging across the split methods. We note that averaging methods are certainly still competitive and discuss this further in Section 6.5 when we study batched updates.

Table 4: Split candidate methods $T = 100, d = 4, Q = 32, \epsilon = 1$

	IH (s=5)	Quantiles	Log	Uniform
Bank	**0.8749**	0.8695	0.8698	0.8734
	(0.0066)	(0.0087)	(0.0087)	(0.0074)
Credit 1	**0.8462**	0.8367	0.8339	0.7822
	(0.0035)	(0.0045)	(0.0058)	(0.0247)
Credit 2	0.7377	0.738	**0.7495**	0.7461
	(0.0084)	(0.0083)	(0.008)	(0.0092)
Adult	**0.8888**	0.8823	0.8848	0.8862
	(0.0035)	(0.0047)	(0.0054)	(0.0034)
Higgs	0.7211	**0.7352**	0.688	0.6449
	(0.0181)	(0.0082)	(0.0141)	(0.0293)
Nomao	**0.9026**	0.8987	0.9003	0.9021
	(0.0041)	(0.0052)	(0.0061)	(0.005)

6.4 Split Candidate Methods

In this section we explore the split candidate methods introduced in Section 4.5. We are interested in comparing the Iterative Hessian (IH) method against the private baseline of uniform splitting and the non-private method of quantiles. We mentioned in Section 4.5 that Log splits are a viable alternative if we know the skew of features. We will assume that we have prior knowledge about skew and thus Log splits have no extra privacy cost. We will show IH can achieve similar or better results than Log splits with the additional benefit that this prior knowledge is not required.

6.4.1 Varying s. One disadvantage of IH splitting when using a TR ensemble is that we must specify the number of split candidate rounds s where budget is spent to produce a Hessian histogram. Figure 3a shows the effect of $s \in \{5, 10, 20, 30, 100\}$ on the Credit 1 dataset with $T = 100$ trees while varying $\epsilon \in [0.1, 1]$ with DP-TR Newton. For higher values of ϵ there is not so much difference between calculating a Hessian histogram for each round ($s = 100$) compared to calculating a Hessian histogram for only $s = 5$ rounds. Although there is a clear trend that on Credit 1 only ~ 5 rounds of IH are needed. As ϵ decreases this difference becomes more apparent. When $\epsilon = 0.5$ we see a 0.04 difference in AUC between $s = 5$ and $s = 100$. At $\epsilon = 0.1$ spreading the already thin privacy budget to compute Hessian information at each tree results in drastically worse performance with $s = 100$. Hence when ϵ is small, spending

more of it on the Hessian histogram results in similar models to using uniform split candidates and we lose the benefits of more informed split candidates.

6.4.2 Comparison of methods. In Figure 3b we fix $s = 5$ for IH and compare the performance on Credit 1 against the other split candidate methods: Uniform, Log, and Quantiles. We vary $T \in \{100, 200, 300\}$ and fix $\epsilon = 1$. Consistently across the different parameter settings the Log splits perform well. This is because Credit 1 contains many skewed features. However, IH with $s = 5$ (our private variant) can indeed match and in some cases exceed Log splits. This indicates that proposing and refining split candidates around (noisy) Hessian histograms is a useful method when datasets have skewed features. We also note that uniform split candidates perform the worst out of all split candidate methods on Credit 1. We also observe here that quantiles (the common choice for non-private boosting methods such as XGBoost) do not lead to the best AUC under DP. In particular, there is a large gap for $T = 200, 300$. Yet for $T = 100$, quantiles perform similarly to Log and IH candidates.

In Appendix A.4 we detail an additional experiment varying ϵ with the split candidate methods. We observe that even if ϵ is small, IH still outperforms the other methods.

In Table 4 we compare the split candidate methods across all the datasets using the same parameter setting. Our IH method shows a clear advantage over uniform on Credit 1 and Higgs where features are particularly skewed. On Credit 2 our IH method achieves the worst performance. However, it does match quantiles in performance. This suggests that quantiles do not produce the best split candidates for Credit 2. It is also likely that because Credit 2 has a large number of categorical features that the repeated splitting in IH serves no additional benefit and could be detrimental to performance. On other datasets none of the features have any notable skew and all split candidate methods perform equally well.

6.4.3 Varying Q. The advantage of the IH method is its more informed split candidates for very skewed features. One may think that we can circumvent the issues of uniform splitting by increasing the number of split candidates, thus considering more fine-grained candidates. In Figure 3c, we fix $T = 100, d = 4, \epsilon = 1$ and vary $Q \in \{2, 4, 8, 16, 32, 64, 128\}$ on Credit 1. The same experiment is detailed on other datasets in Appendix A.5. We can immediately observe further issues with uniform split candidates when combined with TR splits. While proposing more candidates results in fine-grained split choices, the variance from choosing such splits completely at random results in very variable performance when using > 32 split candidates. The experiment supports our choice of $Q = 32$ in other experiments, and also shows that the IH method is relatively robust to the initial number of split candidates.

Summary: We recommend using the IH method to iteratively refine split candidates over a small number of rounds, finding that $s = 5$ usually works the best. Other private split methods like Uniform and Log are competitive depending on the dataset.

6.5 Batched Updates

In Section 5.2 we discussed that boosting is an inherently sequential process and so can take a large number of communication rounds in distributed settings. This is exacerbated by the TR method that often requires a large number of trees (rounds) to achieve good performance. We proposed the idea of batching updates by averaging weights across multiple trees before performing a boosting round. In Figure 4 we vary $\epsilon = 0.1, 0.5$ and fix $T = 200, d = 4$ on the Credit 1 dataset. The same experiment on other datasets is presented in Appendix A.6. We compare the Newton method which takes $T = 200$ rounds and the averaging method which only takes 1 round. We then consider batched updates, varying the size of the batch as $B = p \cdot T$ for $p \in \{0.05, 0.1, 0.25, 0.5, 1\}$.

Focusing first when $\epsilon = 1$ we observe that the Newton model achieves the best performance. This is followed by batched updates that perform some amount of boosting (i.e, $B < 200$). As an example taking $B = 100$ results in only 2 rounds of boosting. A surprising

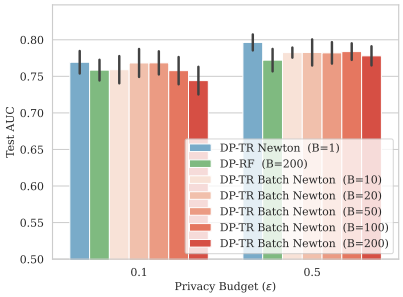


Figure 4: Batched updates, $T = 200, d = 4$

Table 5: Batched updates fixing $T = 200, \epsilon = 0.1, d = 4$.

	Bank	Credit 1	Credit 2	Adult	Nomao
Batch (B=10)	0.7876	0.7585	0.719	0.8438	0.8859
	(0.0233)	(0.0147)	(0.0156)	(0.0086)	(0.0056)
Batch (B=20)	**0.819**	0.7591	**0.7199**	**0.86**	**0.8929**
	(0.0108)	(0.0194)	(0.0164)	(0.0057)	(0.0055)
Batch (B=200)	0.7752	0.7578	0.71	0.8443	0.8858
	(0.0143)	(0.0194)	(0.0146)	(0.0065)	(0.0061)
DP-RF (B=200)	0.7663	0.7441	0.7106	0.8382	0.8852
	(0.0127)	(0.0196)	(0.0106)	(0.0105)	(0.0058)
Newton (B=1)	0.7866	**0.7693**	0.695	0.8371	0.8669
	(0.0224)	(0.016)	(0.0134)	(0.0148)	(0.0088)

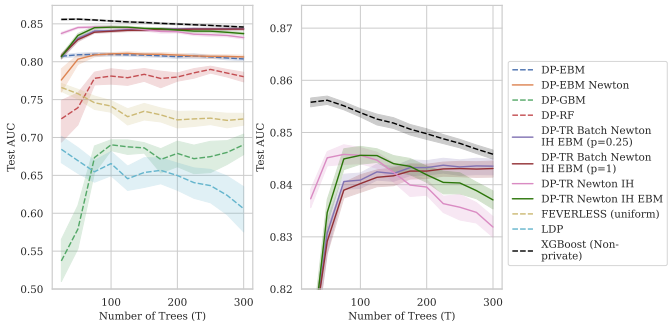


Figure 5: Comparison of DP-GBDT methods and LDP baseline on Credit 1, $d = 4, \epsilon = 1$; Left (all methods), Right (zoomed)

Table 6: Average rank of methods across datasets—rank 1 for highest AUC. (∗) indicates new methods in our framework

	ϵ		
Methods	0.1	0.5	1.0
DP-EBM	5.83	4.5	3.5
DP-EBM Newton (∗)	4.0	3.33	3.17
DP-GBM	9.0	9.0	9.0
DP-RF	4.5	6.67	7.0
DP-TR Batch Newton IH EBM (p=0.25) (∗)	**1.17**	**2.33**	3.33
DP-TR Batch Newton IH EBM (p=1) (∗)	2.0	3.5	4.67
DP-TR Newton IH (∗)	5.33	4.5	3.83
DP-TR Newton IH EBM (∗)	5.17	3.17	**2.67**
FEVERLESS (uniform)	8.0	8.0	7.83

observation is that limiting to 2 rounds of communication achieves a very similar performance to the full Newton model that requires 200 rounds of boosting. When $\epsilon = 0.1$ Newton boosting still performs the best but we observe batched updates with $B = 20, 50$ and thus only a small number of boosting rounds perform very similarly.

To study this more closely, we present a similar experiment in Table 5, fixing $\epsilon = 0.1$ We vary the batch size B and compare to averaging and Newton boosting across all the datasets. We consider TR trees, uniform split candidates, and $T = 200, d = 4$. We still observe that batched updates is a surprisingly competitive alternative to the full boosting procedure across all datasets. We note as in Figure 4 that all methods on Credit 1 are roughly within random variation of one another. The difference in methods is more striking on other datasets with batched updates of size $B = 20$ performing better than Newton. This suggests that under a setting where more noise is added to the training process, boosting is a more unreliable method as it attempts to correct mistakes from previous rounds and can lead to overcompensating for noise. By batching updates we help to average out noise and boost a smoothed update. Generally, batched methods with $B = 20$ or $B = 50$ achieve the best performance with 10 and 5 rounds of boosting respectively. In most cases taking $B = 200$, resulting in a single round of communication (and no actual boosting) only loses at most 0.04 AUC compared to other batched update methods.

Summary. We recommend batching Newton updates to reduce communication rounds and have shown it loses little in performance. Under high privacy, small batches ($p = 0.25 - 0.5$) seem to give the best performance and even beat private Newton boosting.

6.6 End-to-end Comparisons

We conclude with comparisons between baseline methods and those formed from selecting the best options found in previous sections.

Summary across datasets In Table 6 we display the average rank of a method across each of the 6 datasets when ranked in terms of their mean test AUC, where a rank of 1 indicates the highest AUC. We fix $T = 100$ and vary $\epsilon \in \{0.1, 0.5, 1\}$. We observe that most baseline methods underperform and rank consistently in the lower half. The closest competitor DP-EBM performs well when $\epsilon = 1$ but is beaten by DP-TR Newton IH EBM which consistently ranks higher across datasets. When ϵ is small, our batch boosting variant consistently ranks the best across all datasets.

Discussion on Credit 1 To investigate further, we fix $\epsilon = 1, d = 4$ and vary T on Credit 1 in Figure 5. We present comparisons on other datasets in Appendix A.7. These results are best broken down into four main observations which reflect conclusions from previous sections. The first observation is the performance of histogram-based methods. DP-GBM performs the worst followed by FEVERLESS. This shows (as in Section 6.3) that Newton updates when combined with histogram-based methods do increase model performance over normal gradient updates, but in either case, training a large tree with many features entails adding a large amount of noise into the training process and generally poor models.

The second remark concerns the performance of the TR methods. We see a clear performance gap between the DP-RF and DP-TR Newton methods which indicates that boosting does enhance performance when compared to DP-RF. This was confirmed in Section 6.3 where we observed Newton updates under DP generally provided better performance than gradient and averaging updates.

Thirdly, while DP-EBM is very competitive, we can achieve similar AUC by using Newton updates and not training for the full Tm rounds as in [51]. In Figure 5 DP-EBM trains Tm trees, corresponding to $10T$ on Credit 1. Instead our DP-EBM Newton variant uses Newton updates and trains T trees. This shows that we can get the same performance as DP-EBM with far fewer trees when using Newton updates, while reducing communication rounds.

Finally, we note the performance of batched methods when combined with EBM and IH split candidates. We see batched methods with $p = 0.25, 1.0$ essentially match the performance of DP-TR Newton IH and achieve similar performance to the top methods on this dataset. When compared to the full 200 rounds needed for DP-TR Newton IH there is a negligible loss in performance (< 0.01 AUC) but a dramatic reduction in communication rounds.

Summary. By combining the best options in each component (TR, Newton updates, IH, EBM, and batches with $p = 0.25$) we achieve competitive performance that often outperforms our baselines.

7 CONCLUSION

We have proposed a framework for the differentially private training of GBDT models in the federated setting. By evaluating different options at each stage of our framework we have found a dominant approach based on random splits, Newton updates, cyclical training and our iterative Hessian (IH) method. Our approach often outperforms SOTA methods on a range of datasets and results in models close in performance to non-private counterparts. When combined with batching updates, one can train models in only a small number of communication rounds for little loss in performance.

ACKNOWLEDGMENTS

This work is supported by the UKRI Engineering and Physical Sciences Research Council (EPSRC) under grants EP/W523793/1, EP/R007195/1, EP/V056883/1, EP/N510129/1 EP/W037211/1 and EP/S035362/1. This material is also based on work supported by DARPA under agreement number 885000, Air Force Grant FA9550-18-1-0166, ARO with grant W911NF-17-1-0405 and the National Science Foundation (NSF) with grants 1646392, 2039445, CNS-2220433, CNS-2213700, CCF-2217071, CCF-FMitF-1836978, SaTC-Frontiers-1804648 and CCF-1652140.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *ACM SIGSAC conference on computer and communications security*. 308–318.

[2] Naman Agarwal, Peter Kairouz, and Ziyu Liu. 2021. The skellam mechanism for differentially private federated learning. *Advances in Neural Information Processing Systems* 34 (2021).

[3] Vahid R Asadi, Marco L Carmosino, Mohammadmahdi Jahanara, Akbar Rafiey, and Bahar Salamatian. 2022. Private Boosted Decision Trees via Smooth Re-Weighting. *arXiv preprint arXiv:2201.12648* (2022).

[4] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. 2014. Searching for exotic particles in high-energy physics with deep learning. *Nature communications* 5, 1 (2014), 1–9.

[5] Borja Balle and Yu-Xiang Wang. 2018. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *International Conference on Machine Learning*. PMLR, 394–403.

[6] Bank of England. 2019. Machine learning in UK financial services. https://www.bankofengland.co.uk/report/2019/machine-learning-in-uk-financial-services

[7] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure single-server aggregation with (poly) logarithmic overhead. In *ACM SIGSAC Conference on Computer and Communications Security*. 1253–1269.

[8] Jonas Böhler and Florian Kerschbaum. 2020. Secure multi-party computation of differentially private median. In *USENIX Security Symposium*. 2147–2164.

[9] Jonas Böhler and Florian Kerschbaum. 2021. Secure Multi-party Computation of Differentially Private Heavy Hitters. In *ACM SIGSAC Conference on Computer and Communications Security*. 2361–2377.

[10] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.

[11] Philippe Bracke, Anupam Datta, Carsten Jung, and Shayak Sen. 2019. Bank of England: Machine learning explainability in finance: an application to default risk analysis. https://www.bankofengland.co.uk/working-paper/2019/machine-learning-explainability-in-finance-an-application-to-default-risk-analysis

[12] Laurent Candillier and Vincent Lemaire. 2012. Nomao Dataset. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/datasets/nomao

[13] Clément L Canonne, Gautam Kamath, and Thomas Steinke. 2020. The discrete gaussian for differential privacy. *arXiv preprint arXiv:2004.00010* (2020).

[14] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. 2021. Membership Inference Attacks From First Principles. *arXiv preprint arXiv:2112.03570* (2021).

[15] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. 2019. Securely sampling biased coins with applications to differential privacy. In *ACM SIGSAC Conference on Computer and Communications Security*. 603–614.

[16] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 13-17-August-2016 (2016), 785–794. https://doi.org/10.1145/2939672.2939785 arXiv:1603.02754

[17] Kewei Cheng, Tao Fan, Yilun Jin, Yang Liu, Tianjian Chen, Dimitrios Papadopoulos, and Qiang Yang. 2021. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems* (2021).

[18] Kevin Deforth, Marc Desgroseilliers, Nicolas Gama, Mariya Georgieva, Dimitar Jetchev, and Marius Vuille. 2021. XORBoost: Tree boosting in the multiparty computation setting. *Cryptology ePrint Archive* (2021).

[19] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).

[20] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

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

[22] Shereen Elsayed, Daniela Thyssens, Ahmed Rashed, Hadi Samer Jomaa, and Lars Schmidt-Thieme. 2021. Do we really need deep learning models for time series forecasting? *arXiv preprint arXiv:2101.02118* (2021).

[23] Wenjing Fang, Derun Zhao, Jin Tan, Chaochao Chen, Chaofan Yu, Li Wang, Lei Wang, Jun Zhou, and Benyu Zhang. 2021. Large-scale Secure XGB for Vertical Federated Learning. In *ACM International Conference on Information & Knowledge Management*. 443–452.

[24] Zhi Feng, Haoyi Xiong, Chuanyuan Song, Sijia Yang, Baoxin Zhao, Licheng Wang, Zeyu Chen, Shengwen Yang, Liping Liu, and Jun Huan. 2019. Securegbm: Secure multi-party gradient boosting. In *IEEE International Conference on Big Data*. IEEE, 1312–1321.

[25] Sam Fletcher and Md Zahidul Islam. 2015. A Differentially Private Decision Forest. *AusDM* 15 (2015), 99–108.

[26] Sam Fletcher and Md Zahidul Islam. 2015. A differentially private random decision forest using reliable signal-to-noise ratios. In *Australasian joint conference on artificial intelligence*. Springer, 192–203.

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

[28] Jerome H Friedman. 2002. Stochastic gradient boosting. *Computational statistics & data analysis* 38, 4 (2002), 367–378.

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

[30] Sivakanth Gopi, Yin Tat Lee, and Lukas Wutschitz. 2021. Numerical composition of differential privacy. *Advances in Neural Information Processing Systems* 34 (2021).

[31] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* 34 (2021).

[32] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815* (2022).

[33] Nicolas Grislain and Joan Gonzalvez. 2021. DP-XGBoost: Private Machine Learning at Scale. *arXiv preprint arXiv:2110.12770* (2021).

[34] Kaggle. 2012. Give Me Some Credit Competition Dataset. https://www.kaggle.com/competitions/GiveMeSomeCredit/data?select=cs-test.csv

[35] Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu. 2021. Practical and private (deep) learning without sampling or shuffling. *arXiv preprint arXiv:2103.00039* (2021).

[36] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.

[37] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2015. The composition theorem for differential privacy. In *International conference on machine learning*. PMLR, 1376–1385.

[38] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[39] Ronny Kohavi and Barry Becker. 1996. Adult dataset. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/nomao

[40] KPMG. 2020. Fighting Fraud with a Model of Models. https://www.nets.eu/solutions/fraud-and-dispute-services/Documents/Nets-KPMG-Fighting-Fraud-with-a-model-of-models-whitepaper-2020.pdf

[41] Alexey Kurakin, Steve Chien, Shuang Song, Roxana Geambasu, Andreas Terzis, and Abhradeep Thakurta. 2022. Toward Training at ImageNet Scale with Differential Privacy. *arXiv preprint arXiv:2201.12328* (2022).

[42] Andrew Law, Chester Leung, Rishabh Poddar, Raluca Ada Popa, Chenyu Shi, Octavian Sima, Chaofan Yu, Xingmeng Zhang, and Wenting Zheng. 2020. Secure collaborative training and inference for xgboost. In *Workshop on privacy-preserving machine learning in practice*. 21–26.

[43] Nhan Khanh Le, Yang Liu, Quang Minh Nguyen, Qingchen Liu, Fangzhou Liu, Quanwei Cai, and Sandra Hirche. 2021. FedXGBoost: Privacy-Preserving XGBoost for Federated Learning. *arXiv preprint arXiv:2106.10662* (2021).

[44] Darvish Lee Shadravan. 2022. Understanding the Differentiating Capabilities and Unique Features of Salesforce Einstein Discovery within the Machine Learning Space.

[45] Qinbin Li, Zhaomin Wu, Zeyi Wen, and Bingsheng He. 2020. Privacy-preserving gradient boosting decision trees. In *AAAI Conference on Artificial Intelligence*, Vol. 34. 784–791.

[46] Yin Lou, Rich Caruana, and Johannes Gehrke. 2012. Intelligible models for classification and regression. In *ACM SIGKDD international conference on Knowledge discovery and data mining*. 150–158.

[47] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963* (2017).

[48] Xianrui Meng and Joan Feigenbaum. 2020. Privacy-preserving xgboost inference. *arXiv preprint arXiv:2011.04789* (2020).

[49] Ilya Mironov. 2017. Rényi differential privacy. In *IEEE Computer Security Foundations Symposium*. 263–275.

[50] Sérgio Moro, Paulo Cortez, and Paulo Rita. 2014. A data-driven approach to predict the success of bank telemarketing. *Decis. Support Syst.* 62 (2014), 22–31. https://doi.org/10.1016/j.dss.2014.03.001

[51] Harsha Nori, Rich Caruana, Zhiqi Bu, Judy Hanwen Shen, and Janardhan Kulkarni. 2021. Accuracy, interpretability, and differential privacy via explainable boosting. In *International Conference on Machine Learning*. PMLR, 8227–8237.

[52] Santu Rana, Sunil Kumar Gupta, and Svetha Venkatesh. 2015. Differentially private random forest with high utility. In *IEEE International Conference on Data Mining*. 955–960.

[53] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. 2021. Mycelium: Large-Scale Distributed Graph Queries with Differential Privacy. In *ACM Symposium on Operating Systems Principles*. https://doi.org/10.1145/3477132.3483585

[54] Edo Roth, Daniel Noble, Brett Hemenway Falk, and Andreas Haeberlen. 2019. Honeycrisp: Large-scale Differentially Private Aggregation Without a Trusted Core. In *ACM Symposium on Operating Systems Principles*.
[55] Edo Roth, Hengchu Zhang, Andreas Haeberlen, and Benjamin C. Pierce. 2020. Orchard: Differentially Private Analytics at Scale. In *USENIX Symposium on Operating Systems Design and Implementation*.
[56] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. 2020. Crypte: Crypto-assisted differential privacy on untrusted servers. In *ACM SIGMOD International Conference on Management of Data*. 603–619.
[57] Elaine Shi, TH Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. 2016. Privacy-Preserving Aggregation of Time-Series Data. In *NDSS*.
[58] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *IEEE symposium on security and privacy*. IEEE, 3–18.
[59] Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. *Information Fusion* 81 (2022), 84–90.
[60] Pierre Stock, Igor Shilov, Ilya Mironov, and Alexandre Sablayrolles. 2022. Defending against Reconstruction Attacks with Rényi Differential Privacy. *arXiv preprint arXiv:2202.07623* (2022).
[61] Zhihua Tian, Rui Zhang, Xiaoyang Hou, Jian Liu, and Kui Ren. 2020. Federboost: Private federated learning for gbdt. *arXiv preprint arXiv:2011.02796* (2020).
[62] Rui Wang, Oğuzhan Ersoy, Hangyu Zhu, Yaochu Jin, and Kaitai Liang. 2021. FEVERLESS: Fast and Secure Vertical Federated Learning based on XGBoost for Decentralized Labels. (2021).
[63] Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. 2019. Subsampled rényi differential privacy and analytical moments accountant. In *Artificial Intelligence and Statistics*. PMLR, 1226–1235.
[64] I-Cheng Yeh and Che-hui Lien. 2009. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert systems with applications* 36, 2 (2009), 2473–2480.
[65] Lingchen Zhao, Lihao Ni, Shengshan Hu, Yaniiao Chen, Pan Zhou, Fu Xiao, and Libing Wu. 2018. Inprivate digging: Enabling tree-based distributed data mining with differential privacy. In *IEEE INFOCOM*. 2087–2095.

Table 7: Notation

Notation	Meaning
D	Dataset
$\mathcal{T}, T, d$	Forest $\mathcal{T}$ of T trees, each of max depth d
$i \in [n]$	Observation i in the full dataset of n records
$j \in [m]$	Feature index, m features
k	Maximum number of feature interactions
$l \in [L]$	Number of leaf nodes
$q \in [Q]$	Q is the number of split candidates and q is the index
$S_j = \{s_1^j, \dots s_Q^j\}$	s_q^j is q-th split candidate for feature j
$g_i^{(t)}, h_i^{(t)}$	Gradient and hessian of observation i *at the start* of step t
η, β, γ	Regularisation parameters
$\ell(y_i, \hat{y}_i)$	Loss function

Table 8: Datasets

Dataset	n	m	p
Credit 1 [34]	120,269	10	0.07
Credit 2 [64]	30,000	23	0.22
Adult [39]	32,651	14	0.24
Nomao [12]	34,465	10	0.28
Bank Marketing [50]	45,211	16	0.11
Higgs (subset) [4]	200,000	28	0.47

Algorithm 2 Iterative Hessian Splitting

Input: Q - target number of candidates, $H_j^{(t)}$ - Hessian histogram for each feature calculated at the previous tree

for each feature $j \in [m]$ **do**
  $S_j^{(t+1)} = \emptyset, \forall j = 1, \dots, m$
  Compute the target per-bin Hessian $\theta = \frac{\sum_{q \in [Q]} H_{jq}^{(t)}}{Q}$
  for each bin $q \in H_j^{(t)}$ **do**
    if $H_{j,q}^{(t)} < \theta$ **then**
      Merge bin q with bin $q+1$ via $H_{j,q+1}^{(t)} = H_{j,q}^{(t)} + H_{j,q+1}^{(t)}$
    else
      Calculate the bin midpoint $s^* = (s_{q-1}^j + s_q^j)/2$
      Add s_{q-1}^j, s_q^j, s^* to $S_j^{(t+1)}$
  Calculate the number of empty bins $b = Q/|S_j^{(t+1)}|$
  for each $i \in S_j^{(t+1)}$ **do**
    Add b splits to $S_j^{(t+1)}$ uniformly over $[s_{q-1}^j, s_q^j]$
return new split candidates $\{S_1^{(t+1)}, \dots, S_m^{(t+1)}\}$

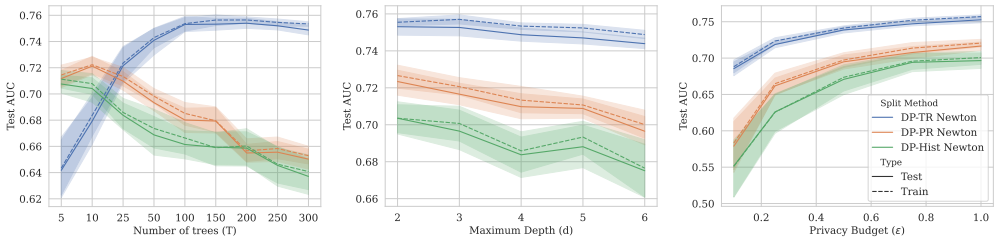
A FURTHER EXPERIMENTS

In this appendix we detail additional experiments that were omitted from the main text. In Section 6 we presented experiments for each component in our framework. To do so, we often showed results on the Credit 1 dataset before varying methods across all of the datasets. This appendix contains plots in the same style of Section 6 but on datasets other than Credit 1. We present the full details of all datasets used in our experiments in Table 8.

A.1 Split Methods: Other Datasets

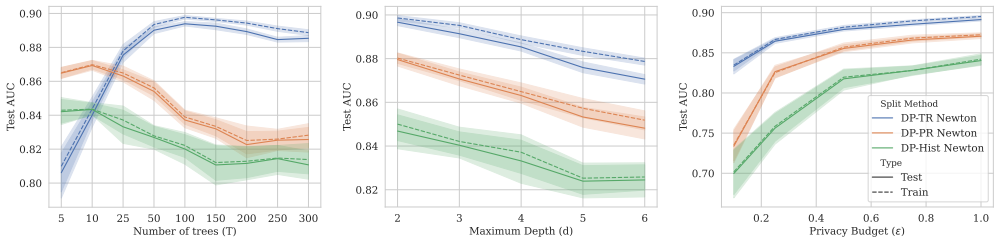
In Section 6.2 we presented Figure 2 which looked at split methods while varying T, d and ϵ on Credit 1. Here we present the same setup but for the Credit 2, Adult, Bank and Nomao datasets. These are presented in Figures 6—9. To summarise, the main conclusions from Section 6.2 also hold across the other datasets. We still observe the same differences when varying T with totally random (TR) splits obtaining the best performance followed by partially random (PR) and then histogram-based (Hist). The gap in AUC performance between Hist and TR is also consistent across the datasets. Furthermore, Hist and PR tend to achieve their best results when T is small (i.e $T \in [10, 50]$) which further justifies our choice of $T = 25$ for Hist/PR in Table 3.

When varying the maximum depth (d) we again find a consistent pattern across the datasets with a performance decrease as we increase the maximum depth. As in Section 6.2 this is likely due to the fact that deeper trees result in leaf nodes with a smaller number of observations and adding noise to this can destroy the weight update information. For some datasets (Adult, Bank and Nomao) there is some evidence of overfitting as d increases which also explains the decrease in performance. Finally, when varying ϵ the trend is again consistent with Section 6.2 and across other datasets. One observation of interest is that for very small ϵ on the Nomao dataset all three split methods perform very similarly.



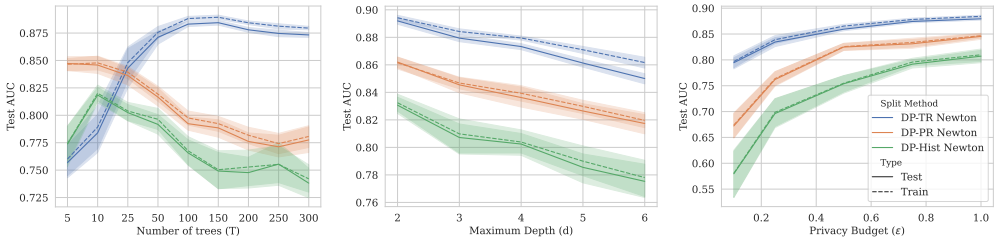
(a) Varying T **(b) Varying D** **(c) Varying ϵ**

Figure 6: Split Methods on Credit 2



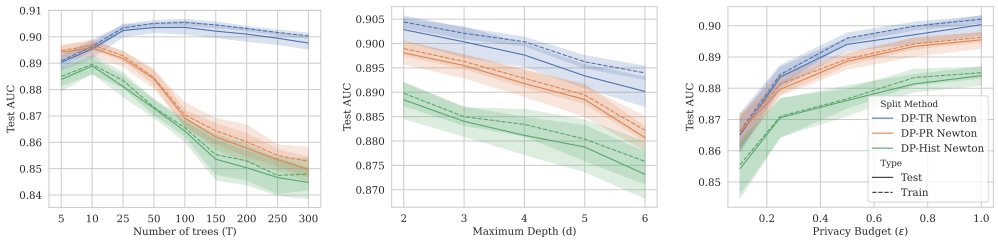
(a) Varying T **(b) Varying D** **(c) Varying ϵ**

Figure 7: Split Methods on Adult



(a) Varying T **(b) Varying D** **(c) Varying ϵ**

Figure 8: Split Methods on Bank



(a) Varying T **(b) Varying D** **(c) Varying ϵ**

Figure 9: Split Methods on Nomao

Table 9: Weight update methods across the datasets fixing $\epsilon = 0.1$

		Bank	Credit 1	Credit 2	adult	nomao
Hist	Gradient	**0.607 +- 0.0698**	0.5415 +- 0.1124	0.5444 +- 0.089	0.6385 +- 0.0892	0.8382 +- 0.0312
	Averaging	0.6002 +- 0.0554	**0.7018 +- 0.0498**	0.5383 +- 0.0596	0.6518 +- 0.0984	**0.8731 +- 0.0141**
	Newton	0.6032 +- 0.0805	0.5879 +- 0.0949	**0.5729 +- 0.0808**	**0.6842 +- 0.0572**	0.8525 +- 0.0264
PR	Gradient	0.5993 +- 0.052	0.5678 +- 0.0909	0.5922 +- 0.074	0.6512 +- 0.0517	0.8504 +- 0.0121
	Averaging	**0.6597 +- 0.0506**	**0.6351 +- 0.0618**	**0.6457 +- 0.0438**	0.7036 +- 0.066	**0.8828 +- 0.0065**
	Newton	0.6413 +- 0.056	0.6264 +- 0.0486	0.5869 +- 0.049	**0.7207 +- 0.0413**	0.86 +- 0.0146
TR	Gradient	0.7413 +- 0.0257	0.7404 +- 0.0296	0.6917 +- 0.017	0.8173 +- 0.0167	0.8743 +- 0.0056
	Averaging	**0.78 +- 0.0181**	0.7669 +- 0.0139	**0.7133 +- 0.0112**	**0.8471 +- 0.0086**	**0.8862 +- 0.0052**
	Newton	0.7742 +- 0.0223	**0.7694 +- 0.015**	0.6921 +- 0.0156	0.8385 +- 0.008	0.8665 +- 0.011

Table 10: Weight update methods across the datasets fixing $\epsilon = 0.25$

		Bank	Credit 1	Credit 2	adult	nomao
Hist	Gradient	0.6505 +- 0.0427	0.5959 +- 0.0695	0.5879 +- 0.0453	0.6927 +- 0.0522	0.8308 +- 0.0319
	Averaging	**0.7078 +- 0.0418**	0.6917 +- 0.0539	**0.6383 +- 0.0343**	0.6631 +- 0.0626	**0.8858 +- 0.0043**
	Newton	0.6878 +- 0.0381	**0.6935 +- 0.0501**	0.6331 +- 0.0476	**0.7547 +- 0.0384**	0.8696 +- 0.0155
PR	Gradient	0.6207 +- 0.0396	0.6273 +- 0.0598	0.6127 +- 0.0562	0.7094 +- 0.0266	0.8645 +- 0.0087
	Averaging	0.7128 +- 0.045	0.6735 +- 0.0551	**0.6811 +- 0.0235**	0.7699 +- 0.0494	**0.8883 +- 0.0055**
	Newton	**0.7436 +- 0.0158**	**0.7356 +- 0.0322**	0.6594 +- 0.0273	**0.8103 +- 0.0183**	0.8706 +- 0.009
TR	Gradient	0.8175 +- 0.0128	0.7698 +- 0.0186	**0.7226 +- 0.0077**	0.8551 +- 0.0085	**0.8917 +- 0.0077**
	Averaging	0.8268 +- 0.0167	0.7742 +- 0.0183	0.7206 +- 0.0132	0.8608 +- 0.0036	0.8871 +- 0.0055
	Newton	**0.8322 +- 0.0095**	**0.7836 +- 0.0109**	0.7212 +- 0.0096	**0.8619 +- 0.0067**	0.8838 +- 0.0056

Table 11: Weight update methods across the datasets fixing $\epsilon = 0.75$

		Bank	Credit 1	Credit 2	adult	nomao
Hist	Gradient	0.6386 +- 0.0502	0.5547 +- 0.0535	0.6242 +- 0.0387	0.6913 +- 0.0252	0.8499 +- 0.0117
	Averaging	0.732 +- 0.0182	0.6398 +- 0.0576	**0.6827 +- 0.0219**	0.6421 +- 0.0265	**0.8869 +- 0.0075**
	Newton	**0.7789 +- 0.0349**	**0.7616 +- 0.0137**	0.6813 +- 0.0232	**0.8264 +- 0.0164**	0.8798 +- 0.0084
PR	Gradient	0.7112 +- 0.0311	0.6921 +- 0.057	0.6381 +- 0.0477	0.7874 +- 0.0233	0.8806 +- 0.006
	Averaging	0.8168 +- 0.0155	0.7583 +- 0.0335	0.6968 +- 0.0262	0.8419 +- 0.0117	**0.8908 +- 0.006**
	Newton	**0.8192 +- 0.0121**	**0.7764 +- 0.016**	**0.7062 +- 0.018**	**0.8558 +- 0.0097**	0.8865 +- 0.0056
TR	Gradient	0.8649 +- 0.0103	0.7871 +- 0.0103	0.7446 +- 0.0108	0.88 +- 0.0049	**0.9008 +- 0.0047**
	Averaging	0.8506 +- 0.0103	0.776 +- 0.0156	0.7311 +- 0.0111	0.8701 +- 0.0031	0.888 +- 0.0052
	Newton	**0.8711 +- 0.0071**	**0.7993 +- 0.0113**	**0.7459 +- 0.0062**	**0.8845 +- 0.0046**	0.8999 +- 0.005

Table 12: Weight update methods across the datasets fixing $\epsilon = 1$

		Bank	Credit 1	Credit 2	adult	nomao
Hist	Gradient	0.6407 +- 0.0538	0.542 +- 0.0525	0.6288 +- 0.0506	0.6803 +- 0.0288	0.8492 +- 0.016
	Averaging	0.7219 +- 0.0275	0.6132 +- 0.0543	0.6831 +- 0.0191	0.6597 +- 0.0294	**0.8881 +- 0.0053**
	Newton	**0.8024 +- 0.0158**	**0.7647 +- 0.0124**	**0.6837 +- 0.0217**	**0.8332 +- 0.0204**	0.8812 +- 0.0079
PR	Gradient	0.7459 +- 0.0295	0.7301 +- 0.0379	0.6426 +- 0.037	0.8196 +- 0.008	0.885 +- 0.0056
	Averaging	0.8247 +- 0.0151	0.7508 +- 0.0521	0.69 +- 0.0218	0.8546 +- 0.0065	**0.8923 +- 0.0059**
	Newton	**0.8362 +- 0.0155**	**0.7855 +- 0.017**	**0.7098 +- 0.0146**	**0.8631 +- 0.0076**	0.8918 +- 0.0055
TR	Gradient	0.8676 +- 0.008	0.7835 +- 0.0141	0.7445 +- 0.0088	0.8855 +- 0.0031	**0.9019 +- 0.0043**
	Averaging	0.8508 +- 0.0116	0.7757 +- 0.0178	0.7296 +- 0.0127	0.869 +- 0.0057	0.8884 +- 0.0054
	Newton	**0.878 +- 0.0047**	**0.8048 +- 0.0065**	**0.7539 +- 0.0076**	**0.8893 +- 0.0033**	0.901 +- 0.0055

A.2 Weight Updates: Varying ϵ

In Section 6.3 we presented Table 3 which varied the choice of weight updates across the datasets fixing $d = 4, \epsilon = 0.5$. In this appendix we also present similar tables but varying the epsilon values $\epsilon \in \{0.1, 0.25, 0.75, 1\}$. These are shown in Tables 9—12.

In Section 6.3 (Table 3) we observed that, for $\epsilon = 0.5$, Newton updates generally performed the best across the split methods although were occasionally beaten by gradient and averaging updates. We make two further observations here. The first is that when we have a larger privacy budget ($\epsilon = 0.75, \epsilon = 1$) and thus less noise the Newton updates more clearly outperform averaging and gradient updates. The second is to observe that under a higher-privacy setting ($\epsilon = 0.1, \epsilon = 0.25$) averaging updates start to outperform both gradient and Newton updates. This implies that under a setting with more noise, averaging the weights across trees is more effective than boosting. This also provides more evidence for our results with batched boosting in Section 6.5 and in Appendix A.6 where we observe a similar effect in high-privacy regimes.

A.3 Feature Interactions

We mentioned in Section 5.1 that cyclical $k = 1$ (i.e, EBM) performed the best and thus we use this setting in our experimental study. In this section we study the value of k and its effect on performance. In Figure 10 we vary T with the TR Newton method, fixing $\epsilon = 1$ and study the cyclical training method with $k \in \{1, 2, 3, 4, 5, m\}$. We note that varying k has no effect on the privacy budget being spent for TR methods. We observe a clear pattern when T is small, that methods which split only on a small subset of features per tree obtain the highest AUC. When $k = 1$, the cyclical method corresponds exactly to the EBM training method. When T gets large there is little difference between the feature interaction methods as all essentially converge to the highest test AUC they can obtain under DP with totally random (TR) splits. We also observed a very similar pattern when studying the random feature interaction method that takes k random features per tree, but omit this plot.

We next look at the effect that cyclical training with $k = 1$ (i.e., EBM) has on model performance when compared to the standard ($k = m$) method which is free to split on any feature when building a tree. Figure 11 shows the result of an experiment fixing $\epsilon = 1$. There is a clear gap between cyclical and non-cyclical methods in AUC performance and a gap between Newton and Gradient updates, but this gap is lessened when using cyclical training. We also observe that cyclical models seem to achieve the highest test AUC in a smaller number of trees than their non-cyclical counterparts. It remains the case that Newton updates provide slightly better performance than GBM updates (as noted for Credit 1 in Section 6.3 and other datasets in Appendix A.2) but this gap narrows considerably when using cyclical training.

A.4 Split Candidates: Varying ϵ on Credit 1

In Section 6.4 we studied the different split candidate methods when using DP-TR Newton. In Figure 12 we present the same methods while varying $\epsilon \in \{0.1, 0.25, 0.5, 0.75, 1\}$ and fixing $T = 100$. As mentioned in the main text we still observe that the IH method performs well across the different choices of ϵ and provides a clear advantage on Credit 1 over uniform and even quantiles for small ϵ.

A.5 Split Candidates: Other Datasets

In Section 6.4 we presented Figure 3c which varied the number of split candidates Q on the Credit 1 dataset. We present similar figures here for Higgs, Credit 2, Adult, Nomao and Bank in Figure 13. These results provide further justification for our choice of $Q = 32$ in our main experiments, since it performs reasonably well across all datasets. Note however, that $Q = 32$ is not the optimal value for each dataset, but does not hinder performance significantly compared to optimal values of Q. See also Appendix C which further shows robustness to the choice of Q.

Figure 13 also provide further evidence of our conclusions in Section 6.4 (more specifically Table 4). For the Higgs dataset we can see clearly the advantage of the IH split candidate method and that as Q increases we approach the (best) accuracy of using quantiles. Recall that the Higgs dataset has a large number of highly skewed features and this is where IH excels. For Nomao, Bank and Adult we observe that the split candidate methods all perform very similarly. As noted in Section 6.4, IH performed poorly on the Credit 2 dataset and we can observe here that as we increase Q it approaches the AUC of quantiles. Hence IH is still capturing the distribution of features well via split candidates, but that quantiles is not optimal for Credit 2.

A.6 Batched Updates: Other Datasets

In Section 6.5 we presented Figure 4 on the Credit 1 dataset when studying the effect of batching updates. Here we present similar figures but for other datasets: Adult, Bank, Credit 2 and Nomao. This is displayed in Figure 14.

In general, we can draw the same conclusions as in Section 6.5. In particular, batching updates are certainly competitive when compared to DP-TR Newton and that it is a useful way to reduce the number of communication rounds needed to train the model. Two new observations are present when we perform this experiment on these other datasets, specifically for $\epsilon = 0.1$. We find DP-RF to be more competitive with DP-TR Newton and also find that batching updates can sometimes perform better than DP-TR Newton. This shows in general that under higher privacy some form of averaging across trees in the ensemble can help with the larger amount of noise that is being added. This again corroborates our insights from Section 6.5 and Appendix A.2.

A.7 Comparisons: Other Datasets

In Section 6.6 we presented an end-to-end comparison of methods in our framework against existing baselines. Figure 5 details these results on the Credit 1 dataset. We present similar experiments here but for other datasets: Higgs, Credit 2, Adult, Bank and Nomao. These are shown in Figures 15—19.

The four main observations detailed in Section 6.6 on the Credit 1 dataset also largely still hold across the different datasets. These can be summarised by noting that the histogram-based methods DP-GBM and FEVERLESS perform the worst with FEVERLESS usually outperforming DP-GBM. This is followed by DP-RF and then DP-TR Newton with IH. This is common across most of the datasets (including Credit 1 as noted in Section 6.6).

What differs slightly across datasets is the best performing methods. For some datasets (like Higgs) DP-TR Newton with IH and

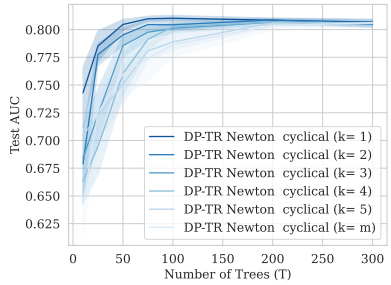


Figure 10: Cyclical k-way on Credit 1

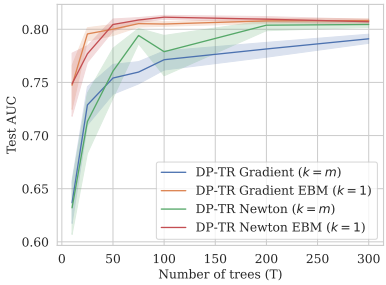


Figure 11: $k = 1$ vs $k = m$ on Credit 1

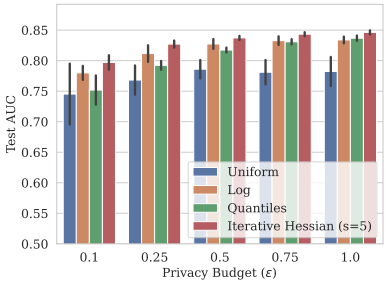


Figure 12: Varying ϵ and split candidate methods on Credit 1 with TR Newton

batched updates clearly perform the best and outperform our baseline methods. On datasets like Credit 2 and Adult where IH split candidates give less of a benefit, the DP-EBM baseline and its variant DP-EBM Newton remain very competitive. The results discussed here are summarised in the rankings presented in Table 6.

B COMPUTATION AND COMMUNICATION

B.1 Communication rounds

The total number of communication rounds depends on the choice of split-method as follows:

- **Histogram-based (Hist).** At every level in the tree, clients must aggregate the required gradient histograms in order to make split decisions for the next level. Hence, the number of communication rounds are of order $O(Td)$, and the size of each message is $O(Qk)$ where $k \leq m$ is the maximum number of features being considered in a tree.
- **Totally random (TR).** For totally random methods, the random trees can be pre-computed by the server and clients are required to aggregate gradient information for leaf nodes of a tree. Thus, the total number of communication rounds is of order $O(T)$, and the size of each message is $O(2^d)$. When using the IH method number of communication rounds is increased to $O(T + s)$ where the message size of each round of IH is $O(Qk)$.
- **Batched updates.** For batched updates with TR, the number of communication rounds is of the order $O(T/B)$, but the messages are B times larger, $O(B2^d)$. If using IH, the number of rounds becomes $O(T/B + s)$.

In the main text, when we refer to batched boosting reducing rounds of communication, this is without considering any secure-aggregation overhead. In practical implementation such as [7], the overhead of secure-aggregation typically requires 3 rounds of communication under an honest-but-curious threat model. Hence, methods like batched boosting are advantageous since reducing a single round of communication results in a 3x reduction under secure-aggregation.

B.2 Communication Cost

The per-round communication cost of methods will vary depending on how the tree is built. We will assume that the server builds a tree a level at a time, which means that gradient information is batched. For example, histogram-based methods at level i will send $O(2^i Qk)$ gradient information corresponding to all gradient histograms at each node in level i. Depending on the federated setting (i.e., availability of devices, communication bandwidth), the server could instead build the tree a node at a time incurring an increase in the number of rounds but a smaller per-round communication cost.

In Figure 20a we present an experiment on synthetic data. This plot is illustrative in the sense that real-world communication costs will be somewhat higher (due to omitting the overheads of secure-aggregation and communication packets). However, the experiment is run on real GBDT models and reveals that the communication cost is often less than the worst-case presented above. For example, many methods that calculate gradient histograms can utilise Hessian information to stop growing the tree early which reduces the overall communication cost.

The experiment in Figure 20a varied over $T \in [25, 300]$, $m \in \{10, 20, 30, 40\}$, $d \in 3, 4, 5$ and $Q \in \{4, 8, 16, 32, 64, 128\}$ as these are the main parameters that effect communication cost. While Figure 20a only displays the total communication cost that needs to be sent over the full training of T trees, the overall per-round communication cost is often much smaller. Even for FEVERLESS while training $T = 1000$ trees results in $\approx$ 100MB communication overall, the per-round communication would be of order ≈ 0.025MB which is reasonable for federated clients (i.e., mobile devices) which may choose to not participate in all rounds. Meanwhile, the methods we advocate incur a total communication cost of under 1MB to build the full model for a client participating in every round.

We note that the total size of data received from the aggregating sever is the same across all methods except for those that use IH, since there is an additional (but small) cost in receiving split candidates over s rounds.

B.3 Computation Cost

Since participating devices in the federated setting are often limited in computation, one may wonder how intensive private GBDT protocols are. In Figures 20c and 20d we provide approximate benchmarks for the client and server costs of methods studied in Section 6.6. These experiments are run without secure-aggregation overheads. We assume a setup where clients have 10,000 samples in their local datasets and we vary parameters $T \in [75, 150], d \in$

$\{3, 4, 5\}, m \in \{10, 20, 30, 40, 50\}, Q \in [16, 32, 64]$ as these all affect the total computation time for the full protocol.

In general, all private GBDT methods are lightweight for both clients and servers. For all methods, clients must compute gradient and Hessian information of their samples. For some methods like FEVERLESS or those that use IH they must also form aggregated gradient histograms. The computation scales linearly in the size of a client's local dataset. For methods that use histograms, clients must also partition their data into Q bins before aggregating.

In practice, the computation overhead of using secure-aggregation would not be large but would dominate most of the local computation costs of the GBDT algorithm. For example, in [7], Bell et al. benchmark client computation costs for $n = 10^5, l = 10^5$ where l is the dimension of the quantity the server is aggregating. Their secure-aggregation algorithm takes ≈ 0.35 seconds for clients. In the worst-case their parameter settings match ours and the per-round computation overhead for clients when using secure-aggregation will be no more than a second under our honest-but-curious threat model.

C VARYING THE NUMBER OF CLIENTS

One may ask whether the private GBDT protocols in our framework are sensitive to the number of participating clients. Every method in our framework relies on aggregating gradient information of the form $q(I) = (\sum_{i \in I} g_i, \sum_{i \in I} h_i)$ from a number of clients, where I is the set of samples at a (leaf or internal) node. Thus, the total number of clients is somewhat unimportant to the overall utility of the algorithm. Instead, the most important factor is the total number of data samples. In our experiments in Section 6 we assumed that one client holds one data item, and hence varying the number of clients is equivalent to varying the number of samples. But in scenarios where each client may hold more than one data item, the number of participating clients is only important if they are providing more data to the algorithm. In practice, we would hope that there is enough participating data to ensure that the aggregated gradient information will still be meaningful under noise.

The number of samples at a given node affects both the construction of the tree (internal splits and leaf weights) and our IH split candidate method as it depends on Hessian histograms. Thus we are interested in analysing two effects: First, how sensitive our methods are to the number of participating clients (when compared to non-private XGBoost) and secondly, the sensitivity of the number of split candidates Q. In particular, we are interested in how sensitive the IH method is to the value of Q since it relies on refining split candidates around Hessian information, which is itself dependent on the total number of participating samples.

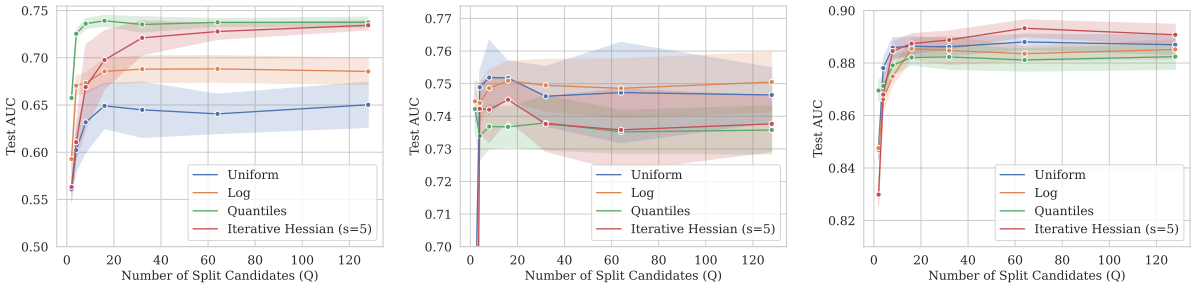
To understand how this affects utility we run an experiment, shown in Figure 20b on synthetic data generated with $m = 30$ and the number of clients (equivalently, the number of samples) $n \in [70, 200000]$[7]. We choose to use a synthetic dataset which is "easy" to classify in the sense that in the non-private setting we only need $n \approx 1000$ samples to get almost perfect (> 0.95) AUC. This allows us to see more clearly the effect that DP noise has when varying the number of clients. We compare our two most competitive methods DP-TR Newton with uniform splits and DP-TR Newton with IH splits while varying n and the number of split candidates Q. We fix $d = 4, \epsilon = 1$.

We observe that when $n < 2000$ the results of our DP methods are poor and the performance gap between non-private XGBoost and our private DP-GBDT methods is large. We find that when $n \geq 10,000$ the DP methods stabilise and achieve a reasonable AUC of around 0.95. This is suitable for the federated setting where $n \geq 10,000$ is expected.

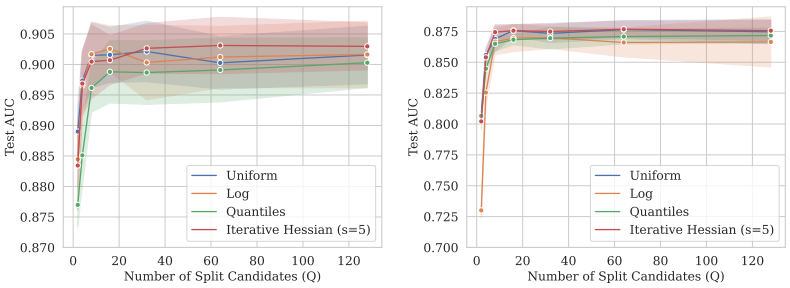
For the number of split candidates Q, we first note that this is not a particularly sensitive parameter in the non-private setting where the only significant gap in performance occurs when $Q = 2$. For the DP methods, the value of Q is understandably more sensitive. We remark that DP-TR Newton (uniform) and DP-TR Newton IH perform very similarly when $n \leq 10,000$ and when $n \geq 10,000$ there is a large gap in performance as long as Q is not too small. This is expected, as when n is small, the Hessian histograms will likely have more noise than signal and thus the split candidates formed from IH are likely to be very similar to uniform. When n is larger, the IH method has more accurate Hessian histograms and so more refined split candidates and thus better performance.

We find in this experiment that values of $Q = 2, 4, 8$ usually result in the lowest AUC (by at most 0.05) and that values between $Q = 32 - 512$ perform very similarly. The general conclusion here is that the choice of Q is robust as long as you do not choose it too small. This also agrees with our conclusions from Section 6.4 and Appendix A.5.

[7] See https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html

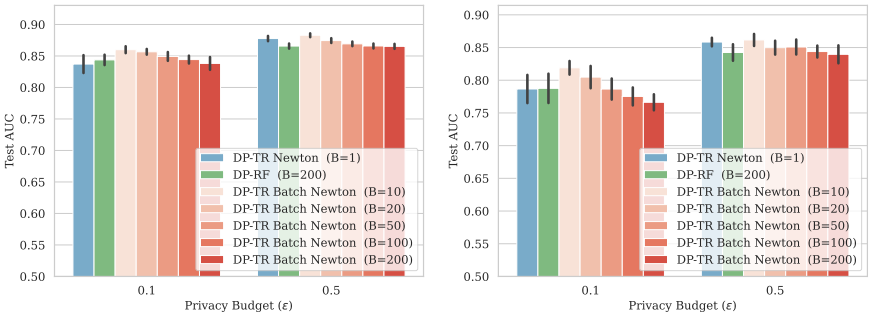


(a) Higgs (b) Credit 2 (c) Adult

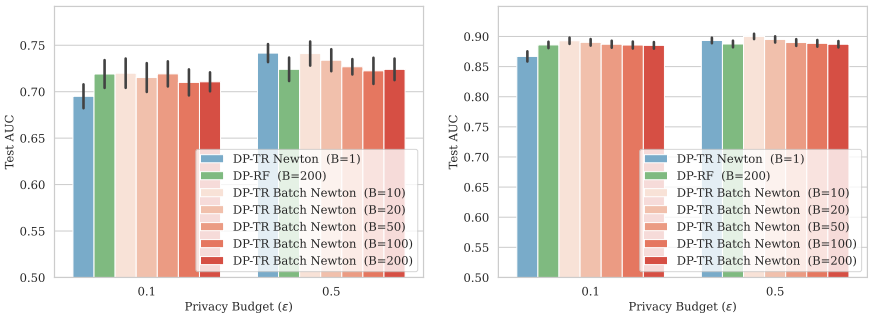


(d) Nomao (e) Bank

Figure 13: Split Candidate Methods: Varying $Q \in \{2, 4, , 16, 32, 64, 128\}$



(a) Adult (b) Bank



(c) Credit 2 (d) Nomao

Figure 14: Batched updates across the datasets ($T = 200, d = 4$)

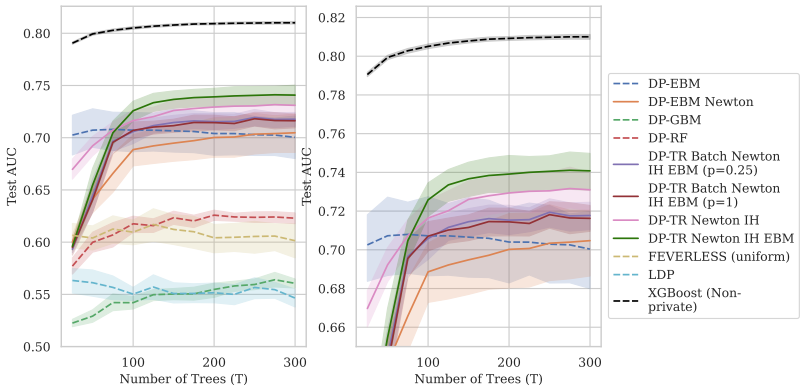


Figure 15: Comparison of methods: Higgs ($d = 4, \epsilon = 1$)

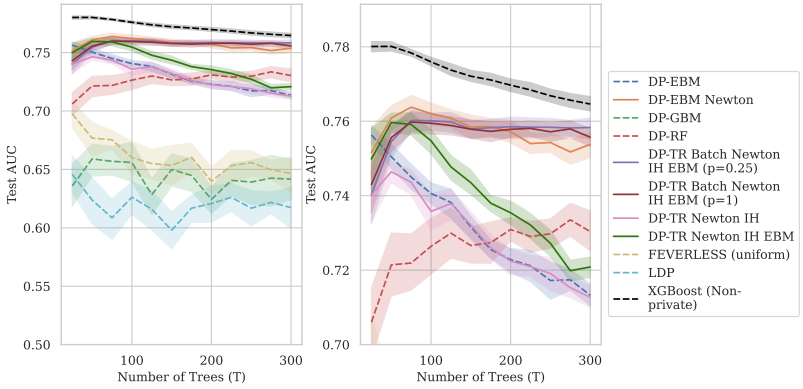


Figure 16: Comparison of methods: Credit 2 ($d = 4, \epsilon = 1$)

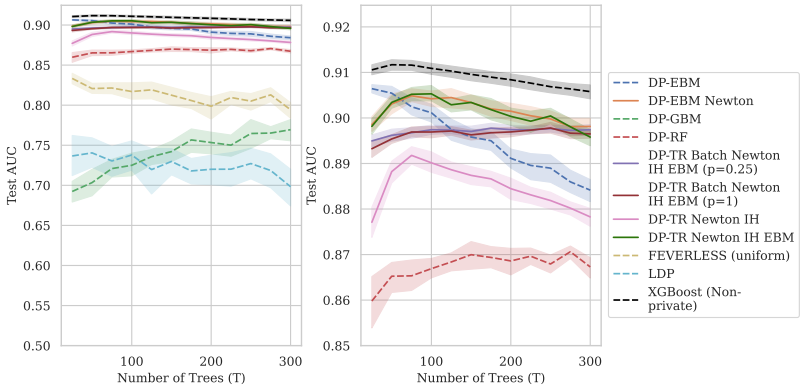


Figure 17: Comparison of methods: Adult ($d = 4, \epsilon = 1$)

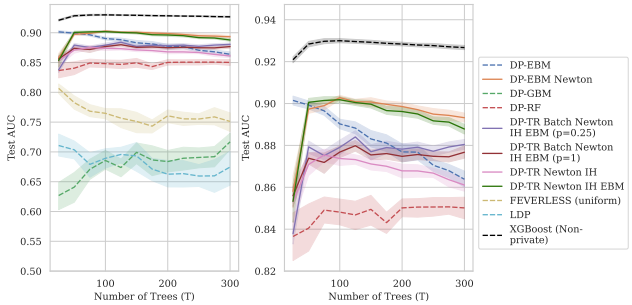


Figure 18: Comparison of methods: Bank $(d = 4, \epsilon = 1)$

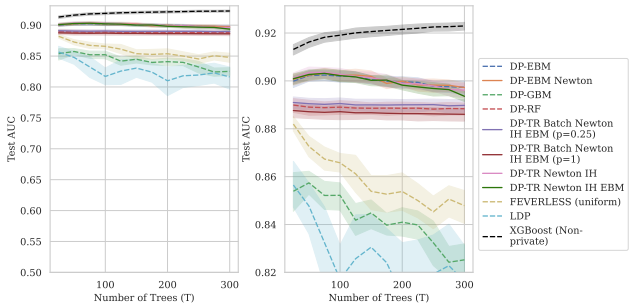
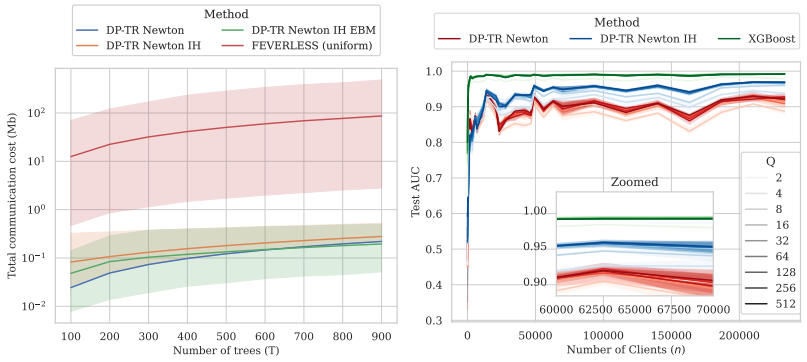
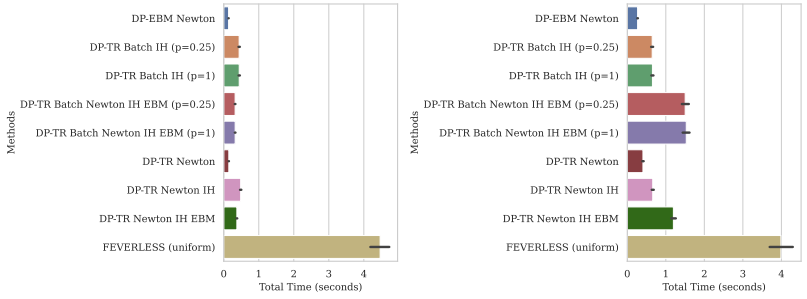


Figure 19: Comparison of methods: Nomao $(d = 4, \epsilon = 1)$



(a) Communication - $d \in [3, 4, 5], m \in [10, 20, 30, 40]$

(b) Varying clients



(c) Client computation

(d) Server computation

Figure 20: Computation Benchmarks