MammothDP: Differentially Private Boosted Decision Trees, hyperparameter-free and ready for Trusted Hardware

Abstract

Machine learning on sensitive data requires a secure processing environment and a privacy-preserving publishing method. A prominent example is the European Health Data Space which shall contain medical patient data of all EU citizens and requires privacy-preservation for published studies.

This work makes two contributions towards such a system using differential privacy (DP) within a trusted execution environment (TEE). We introduce *MammothDP*: a practical and hardened hyperparameter-free DP tree ensemble learner – a system for understanding complex correlations in tabular data. First, prior work results in an increase in the privacy budget of a factor of 2 to 3 for DP hyperparameter-tuning. We experimentally show that using early stopping to decide the number of trees and fixing all other hyperparameters to reasonable values can replace hyperparameter-tuning. We introduce a postprocessing-based early stopping criterion that works for any boosted tree ensemble learners, without any privacy overhead. Second, we tackle the problem of hardening a DP tree ensemble learner by providing a constant-time implementation. We successfully evaluated the implementation with the side-channel leakage tool Microwalk. Moreover, we identify subtle security challenges related to TEE rollback attacks and explain how to avoid them. We experimentally compare MammothDP to the state-of-the-art tree ensemble learner S-BDT. MammothDP achieves significantly stronger results if S-BDT abstains from any hyperparameter-tuning and competitive results if S-BDT conducts a non-private hyperparameter-tuning.

1 Introduction

The real-world deployment of theoretical differentially private (DP) learning algorithms for highly sensitive data comes with challenges that are neglected by the DP literature. A prominent example of a real-world system that requires strong protection is the European health data space (EHDS), which aims to make medical EU-wide patient data available for medical research studies. The EHDS requires studies on the raw sensitive data to be conducted inside a *secure processing environment* [19, Art 50] and then to be published in an *anonymized manner* [19, Art 46(11)]. The state-of-the-art privacy solution for publishing studies in an *anonymized manner* is differential privacy (DP), and a state-of-the-art enforcement method of a *secure processing environment* are hardware-supported trusted execution environments (TEEs), e.g., Intel TDX [34] and SGX [20]. While a TEE protects the data against privileged attackers (OS, hypervisor, administrator) during the data processing, DP protects the data after the processing from any curious 3rd party.

Concerning medical applications, the differentially private method for conducting and publishing the study has to provide strong utility-privacy tradeoffs for medium-sized data sets and explainability. For tabular data, a common task in medical applications, there is a successful line of research on DP tree ensemble learners [7,41,44,53] that meets these requirements and additionally captures complex correlations in the data. The state-of-the-art boosted decision tree ensemble learner S-BDT [53] has displayed strong utility-privacy tradeoffs, even for strict privacy requirements ($\varepsilon \leq 0.1$) and mid-sized dataset sizes (roughly 48k data points).

S-BDT, however, leaves open how to find suitable hyperparameters, such as the number of trees, for the training in a data-independent manner. Prior results only display the utility-privacy tradeoff after a thorough hyperparameter-tuning that is specific to the sensitive data set. While there are DP hyperparameter-selection algorithms [1, 16, 25, 43, 51], these incur a high privacy-penalty (factor 2 to 3 in terms of ε) and deteriorate the utility-privacy tradeoff. While statically fixing all hyperparameters before training [1, 35] has the risk of being too uncertain for a good utility, we raise the question of a sweet spot in between a privacy-penalty and static hyperparameters that we tackle: *Can a hyperparameter-free algorithm work that adapts a hyperparameter like the number of trees during training without spending extra privacy budget, i.e., by solely relying on post-processing?*

Beyond challenges on the algorithmic side, the literature

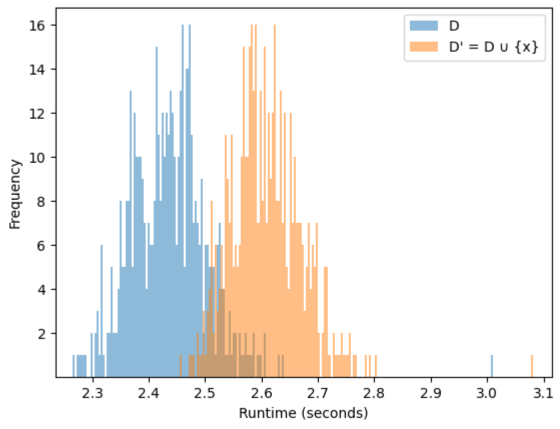


Figure 1: **A single training example can have a measurable impact on the runtime of an S-BDT model although ($\varepsilon = 0.1$)-DP holds**: here, the adversarial advantage is TPR=0.29@FPR=0.002. We train 500 S-BDT models on random subsets of size 3 of Abalone [48] and 500 models with an additional data point x and plot the runtime. S-BDT uses $T = 30{,}000$ trees of depth $d = 8$. The attacker chooses threshold $\tau = 2.64$ to differentiate the models with and without x. The metric TPR@FPR=0.002 quantifies the true positive rate (TPR) at a very low false positive rate (FPR) [14].

has shown that TEEs such as Intel's TDX and SGX still leak side-channel information about the programs and the data that they process, such as timing leakage and memory access patterns. These side-channels are aggravated, as machine learning algorithms, such as S-BDT, conduct a series of complex operations. Figure 1 shows a simple attack on S-BDT that exploits the running time and successfully extracts information about sensitive data, even though strong differential privacy guarantees hold ($\varepsilon = 0.1$). In addition, TEE rollbacks attacks [45, 52] can violate DP guarantees, e.g., by reverting the TEEs state to a previous version and enabling more queries than a privacy budget should allow. While rollback attack mitigations have been proposed in several previous works, our analysis shows that state-of-the-art solutions [36] have limitations and preventing rollback attacks is non-trivial, especially for randomized computations like DP algorithms that also require side-channel protection.

Our Contribution. We introduce *MammothDP*, a constant-time hyperparameter-free DP tree ensemble learner, based on the state-of-the-art S-BDT [53]. On the one hand, we introduce a novel overhead-free method for achieving hyperparameter-freeness for tree ensemble learners like S-BDT. On the other hand, we tackle the above-mentioned deployment challenges.

1. Towards hyperparameter-freeness, we introduce a novel post-processing-based early tree stopping technique for S-BDT that achieves strong utility-privacy tradeoffs without relying on an extra privacy budget and that might be of independent interest. In early tree stopping, we employ a heuristic that halts the training if we detect a possible point of bias-freeness. Thus, this method dynamically chooses the number of training trees. We provide a theoretical foundation for the early stopping technique for regression utilizing the well-known division of the mean squared error (MSE) in a bias and a variance error with the goal of minimizing the bias. We experimentally show that we still perform worse than the privacy-leaky hyperparameter-tuned S-BDT but significantly outperform S-BDT with a statically chosen number of trees.

2. To counter timing-attacks such as Figure 1, we provide a constant time implementation of S-BDT.[1] We harden our system against

   - differing memory access patterns and control flow via adapting S-BDT to use bitmasking and to ensure that we always loop across all data points,
   - differing runtime of instructions via adapting the C++ library *libfixedtimefixedpoint* [3] to S-BDT,
   - differing noise sample runtimes via adapting FACCT [69] to the Gaussian and Bernoulli noise used in S-BDT, and
   - differing data sizes within the threat model DP that lead to a differing number of loop iterations via guaranteeing timing privacy [56].

   We validate our constant-time results using Microwalk [65] and measure the increased computation time. Finally, we identify subtle security challenges in the design of TEE rollback mitigation for computations like DP algorithms and outline a rollback defense that improves the protection of sensitive data over state-of-the-art solutions like [36].

2 Preliminaries

2.1 Trusted Execution Environments

Various TEE technologies exist. TEEs such as Intel's SGX [20] and Arm TrustZone [54] are designed to isolate applications, called enclaves, from the rest of the system including the untrusted OS. Intel's TDX [18] and Arm CCA [42] isolate entire VMs from any untrusted code running on the same physical computing platform. All such TEEs have the same high-level goal: to allow confidential workloads to be executed on a computing platform that is trusted as little as possible (e.g., the OS and administrator can be malicious).

Most TEEs offer two main security guarantees. The first is *data confidentiality*, which is achieved by creating an isolated

[1]https://drive.google.com/1TW8-TDX-image

execution environment. For example, in the case of Intel's SGX, the processor keeps track of the currently executing code and allows confidential data to be accessed only when the correct TEE application is running. The second main security guarantee, *execution integrity*, is achieved through remote attestation, which allows third parties to verify that the expected code is running inside TEE, before providing confidential data to it. These guarantees provide an attractive alternative to secure computation using cryptography, as TEEs can process confidential data efficiently in plaintext without the need of a trusted party [55].

2.2 Differential Privacy

Differential privacy (DP) [24] is the de facto standard for provable privacy. Given a mechanism that has a data-dependent output, DP requires that the impact of single data points in the output is limited and thus deniable. In line with S-BDT [53], we consider unbounded DP, where the effect of adding or removing a single instance from a dataset on the output is analyzed.

Definition 1 (Neighboring datasets)**.** Given two datasets $X \subseteq \mathcal{X}$ and $X' \subseteq \mathcal{X}$ where $X' := X \cup \{x\}$ for some $x \in \mathcal{X}$, then X and X' are neighboring: $X \sim_x X'$, or in short $X \sim X'$.

Definition 2 (Differential Privacy, [24])**.** A randomized mechanism $M : \mathcal{X} \mapsto \mathcal{R}$ satisfies (ε,δ)-DP if, for any two neighboring datasets $X \sim X'$ and any observation o: $\Pr[M(X) = o] \leq e^{\varepsilon} \cdot \Pr[M(X') = o] + \delta$.

S-BDT uses a variant of DP: $(\alpha,\rho(\alpha))$-Rényi DP (RDP) which bounds the α-th moment of the privacy loss.

Definition 3 (Rényi Divergence, [57])**.** Given two probability distributions P,Q over $\mathcal{R}$ where $P(o)$ denotes the density of P at o, then the Rényi divergence of order α is defined as

$$D_\alpha(P||Q) := \tfrac{1}{\alpha-1} \log \int_{-\infty}^{\infty} \tfrac{P(o)^\alpha}{Q(o)^{\alpha-1}} do.$$

Definition 4 (Rényi DP, Def. 4 in [46])**.** A randomized mechanism M: $\mathcal{X} \mapsto \mathcal{R}$ satisfies $(\alpha,\rho(\alpha))$-RDP if for any two neighboring datasets $X \sim X'$: $D_\alpha(M(X)||M(X') \leq \rho(\alpha)$.

With RDP, we can mathematically describe features like composition, filter [27], or subsampling (cf. Theorem 8). With Corollary 5, we can convert RDP to DP. For a tight conversion, we refer to [63, Theorem 2] which requires access to all α.

Corollary 5 (RDP to DP, Thm. 21 in [5] or Prop. 12 in [13])**.** *For any $\delta \in [0,1]$, if a mechanism is $(\alpha,\rho(\alpha))$-RDP, then it is (ε,δ)-DP with $\varepsilon = \rho(\alpha) + \log(\frac{\alpha-1}{\alpha}) - \frac{\log(\delta)+\log(\alpha)}{\alpha-1}$.*

2.3 Timing Privacy

Timing Privacy (cf. Definition 6) is a framework for ensuring DP in the presence of timing side-channels. Intuitively, timing private mechanisms should not leak much more information about their input than what is already revealed by their output distribution.

Definition 6 (Timing Privacy, special case of Definition 31 in [56])**.** A randomized mechanism $M : \mathcal{X} \times \mathcal{E} \mapsto \mathcal{R} \times \mathcal{E}$ satisfies $(1 \mapsto \rho(\alpha))$-Timing Privacy if, for any two neighboring datasets $X \sim X'$, any env, env$' \in \mathcal{E}$ and any $y \in \text{supp}(\text{out}(M(X,\text{env}))) \cap \text{supp}(\text{out}(M(X',\text{env}')))$:

$$D_\alpha(T_M(X,\text{env})|_{\text{out}(M(X,\text{env}))=y}||$$
$$T_M(X',\text{env}')|_{\text{out}(M(X',\text{env}'))=y}) \leq \rho(\alpha)$$

where $T_M(X,\text{env})$ is the runtime of M on inputs X & runtime environment env and $\text{out}(M(X,\text{env}))$ is the output of M.

2.4 Gradient Boosted Decision Trees

Gradient boosted decision trees (GBDT) [28] learn a sequence of decision trees by iteratively correcting errors of prior trees. Let $X = \{(x_1,y_1),\ldots,(x_n,y_n)\} \subseteq \mathbb{R}^m \times \mathbb{R}$ denote a labeled dataset with n data points and m features. For simplicity, we denote y as a label although a regression target applies equally. A tree ensemble model $\phi^{(T)} := [f^{(1)},\ldots,f^{(T)}]$ minimizes

$$\mathcal{L}^{(T)}(\phi,X) = \textstyle\sum_{(x_i,y_i)\in X} l(\phi^{(T)}(x_i),y_i) + \sum_{t=1}^{T} \Omega(f^{(t)})$$

where l is a twice-differentiable convex loss function, e.g. squared error or binary cross-entropy, that measures the difference between the prediction $\tilde{y}_i := \phi^{(T)}(x_i)$ and the label y_i, and $\Omega(f^{(t)}) = {}^1\!/_{2}\lambda||V^{(t)}||^2$ is a regularization term on the leaves vector $V^{(t)} = \text{Leaves}(f^{(t)})$.

For training each tree f_t, XGBoost [17] proposes Newton boosting, a second-order approximation of the loss function:

$$\begin{aligned} l(\phi^{(t)}(x_i),y_i) &= l(\phi^{(t-1)}(x_i) + f^{(t)}(x_i),y_i) \\ &\approx l(\phi^{(t-1)}(x_i),y_i) + g^{(t)}(\phi^{(t)}(x_i),y_i) f^{(t)}(x_i) \\ &\quad + \tfrac{1}{2} h^{(t)}(\phi^{(t)}(x_i),y_i) {f^{(t)}}^2(x_i) \end{aligned}$$

with gradient $g^{(t)}(\tilde{y}_i,y_i) = \partial/\partial\tilde{y}_i\, l(\tilde{y}_i,y_i)$ and Hessian $h^{(t)}(\tilde{y}_i,y_i) = \partial^2/\partial\tilde{y}_i^2\, l(\tilde{y}_i,y_i)$.

Each tree $f^{(t)}$ is recursively built from a root node to the leaves: each node splits a dataset X in a left I_L and right child I_R given some split criterion s. Each child with its remaining dataset is then the basis for the next subtree. The process stops until a stopping criterion, e.g. the maximal tree depth, is reached. The error correction terms of those data points that end up in a leaf are used to construct that leaf.

Optimal leaf value. The leaves of $f^{(t)}$ contain the tree's prediction, which is derived with the Newton method as:

$$V^{(t)}(I_{Leaf}) = -\frac{\sum_{(x_i,y_i)\in I_{\text{Leaf}}} g^{(t)}(\phi^{(t)}(x_i),y_i)}{\sum_{(x_i,y_i)\in I_{\text{Leaf}}} h^{(t)}(\phi^{(t)}(x_i),y_i)+\lambda} \tag{1}$$

for a subset I_{Leaf} of training data X that ended up in this leaf.

2.5 S-BDT

S-BDT is the state-of-the-art (ε,δ)-differentially private distributed GBDT learner that shows strong potential for learning tabular data even for smaller data sets ($< 5K$ data points) and vigorous privacy guarantees ($\varepsilon < 0.5$).

S-BDT proposes a DP initial score as the initial classifier that outputs the mean of labels from the dataset released via the Laplace mechanism to preserve $(\alpha,\rho(\alpha))$-Rényi DP.

A leaf is constructed using Newton boosting with all data within a leaf (cf. Algorithm 7). The leaf value is DP approximated using leaf-balanced noise [53, Theorem 16], a non-spherical variant of the multivariate Gaussian mechanism.

S-BDT utilizes randomized splits that can yield good utility [7, 44, 53] and limits the privacy leakage to learning leaf-values. Random splits are constructed by randomly selecting a feature of the dataset and a value for that feature as a split.

S-BDT enables privacy amplification by subsampling for every iteration of tree training in the ensemble. The amplified RDP bound of tree training is given in Theorem 7 and uses the bound of Theorem 8.

Theorem 7 (Individual RDP bound of subsampled tree training, Theorem 19 in [53])**.** *Let $r_d \in \mathbb{R}^+$ $(d = 1,2,...,D)$ such that $\sum_{d=1}^{D} r_d = 1$. Let σ^2_{leaf} be the unweighted variance of the leaf Gaussian. Let $a_\gamma : \mathbb{N} \times \mathbb{R} \mapsto \mathbb{R}$ denote the privacy amplification of Theorem 8 with subsampling ratio γ. Then, for data point x_i with gradient g_i and Hessian h_i, Algorithm 6 (`TrainSingleTree`) satisfies $(\alpha, a_\gamma(\alpha, \alpha \cdot 2/2\sigma^2_{leaf} \cdot \left(\frac{r_1 \cdot |h_i|^2}{(h^*)^2} + \frac{r_2 \cdot |g_i|^2}{(g^*)^2}\right)))$-individual RDP.*

Theorem 8 (Privacy Amplification by Subsampling, Theorem 8 in [70])**.** *Let $\mathcal{M}$ be any randomized mechanism that obeys $(\alpha,\rho'(\alpha))$-Rényi differential privacy. Let γ be the subsampling ratio and $\alpha \geq 2$. Let $M^{\mathcal{P}_\gamma} = \mathcal{M} \circ \mathcal{P}_\gamma$ and $\mathcal{P}_\gamma$ generating a Poisson subsample with subsampling ratio γ. If for all neighboring datasets $X \sim X'$ and all odd $3 \leq l \leq \alpha$, $D_{\chi^l}(M(X)||M(X')) \geq 0$ then $M^{\mathcal{P}_\gamma}$ is tightly $(\alpha,\rho(\alpha))$-Rényi differentially private with $\rho(\alpha) = \frac{1}{\alpha-1} \log\left((1-\gamma)^{\alpha-1}(\alpha\gamma - \gamma + 1) + \sum_{l=2}^{\alpha} \binom{\alpha}{l} (1-\gamma)^{\alpha-l} \gamma^l e^{(l-1)\cdot\rho'(l)}\right)$.*

S-BDT tailors individual privacy accounting via a Rényi filter [27] to BDT training which allows training for an arbitrary amount of training rounds when filtering out in every round the data points that have exceeded their individual privacy budget. The individual privacy budget is updated using the individual RDP bound (cf. Theorem 7) and compared against the upper bound on the privacy budget.

2.6 Timing side-channel & constant-time code

Timing side-channel attacks exploit information leakage through data-dependent timing behavior of algorithms. These attacks exploit shared hardware features like caches [9, 21, 30, 31, 47, 60] and branch prediction units [2, 39], they leak information to a malicious hypervisor using the page table [12, 68] or by precisely tracking the control flow of a TEE using single-stepping [11, 67]. The most widely-used protection against timing side-channels is software hardening using constant-time programming techniques [10, 32, 38, 50]. Constant-time programming implies using microarchitectural resources like caches in a way that is independent of sensitive data.

Microwalk Code written in a constant-time fashion does not necessarily guarantee security against timing attackers. For example, subtle leakages may be missed [61, 64], or compiler optimizations may reintroduce vulnerabilities [22, 62]. Thus, verification of compiled binaries is needed. There has been much research in automated constant-time verification and many tools were proposed [29]. One of these tools is Microwalk [65, 66], a dynamic side-channel analysis framework which has since its initial proposal matured to a stable and practically usable tool.

Microwalk uses a dynamic approach, i.e., it analyzes a program for a concrete set of random inputs, which are carefully crafted to achieve high implementation coverage. For every such input, Microwalk collects an execution trace with all taken branches and accessed memory addresses, which resembles an overapproximation of the information a powerful side-channel attacker can acquire. The subsequent side-channel leakage analysis follows a simple idea: If all these traces are *identical*, the random inputs led to the same execution path, i.e., they are indistinguishable for a side-channel attacker. On the other hand, if there are *differences* between these traces, the attacker can distinguish the inputs and thus learn (partial) information. Microwalk compares execution traces and computes a leakage score, indicating how much information a side-channel attacker would get in the worst case. To aid fixing the identified vulnerabilities, Microwalk outputs information about the source code location and witness inputs that trigger the vulnerability.

libfixedtimefixedpoint library The libfixedtimefixedpoint (libftfp) library [3] is a fixed-point constant-time math library. It uses a floating point representation with a fixed number of bits for the integer and fractional portion and implements all floating point operations using only integer instructions that are constant-time.

FACCT discrete Gaussian sampling FACCT [69] is a C++ library for sampling from a discrete Gaussian. For a discrete Gaussian D_σ with $\sigma = k \cdot \sigma_0$ and $\sigma_0 = \sqrt{1/(2\ln 2)}$, FACCT uses the binary sampling method of the BLISS signature scheme [23]: First call the base sampler to generate a sample $x \sim \mathcal{D}^+_{\sigma_0}$ from a positive discrete Gaussian with standard deviation σ_0. The base sampler uses an adaptation of the inversion method from Bos et al. [8]. The inversion method samples $x \sim \text{pdf}_f$ from a probability density function pdf_f by sampling $u \sim \mathcal{U}([0,1])$ and then computing $F_f^{-1}(x)$ with the inverse CDF F_f^{-1}. This approach can be implemented in constant-time using a full-table access CDT sampler.

Now, generate $y \sim \mathcal{U}(\{0,1,...,k-1\})$, set $z := kx + y$, $t := y(y+2kx)$ and perform rejection sampling on z with rejection rate $p = \exp\left(-y(y+2kx)/(2\sigma^2)\right)$ using a Bernoulli sampler. The Bernoulli sampler needs to compute p first to generate a sample. Zhao et al. [69] propose to compute a polynom P as an approximation of 2^x for $0 \leq x < 1$ which is then further used to compute p (cf. [69, Figure 8]) . Zhao et al. find a polynom P that has maximal relative error $\max_x |\frac{P(x)-2^x}{2^x}| < 2^{-45}$. They evalute P which has degree n and coefficients $(a_i)_{i=1}^n$ by using Horner's rule $(H(x))$ with δ_{FP}-bit precision, yielding absolute error $|P(x) - H(x)| \leq \gamma_{2n} \cdot \sum_{i=0}^n |a_i| \cdot |x|^i$, where $\gamma_{2n} \approx 2n \cdot 2^{-\delta_{\text{FP}}}$. Evaluating the polynom is possible in constant-time because the degree and precision is fixed.

The probability of outputting an integer $z = kx + y$ is now proportional to $\exp\left(-z^2/(2\sigma^2)\right)$ ([69, Theorem 3]).

3 Hyperparameter-free: Early Tree Stopping

When making differentially private machine learning algorithms like S-BDT [53] robust against OS-level attackers inside a TEE, we encounter hyperparameter tuning. Hyperparameter tuning retrains an ML model many times to select the hyperparameters with the highest score (e.g., RMSE, AUC, etc.). This is a privacy issue but also an issue predominant for OS-level attackers: since all operations and data are solely within a TEE, no developer can easily debug, test, or evaluate the utility of the ML algorithm before releasing the final differentially private model. Hence, these privacy and practical considerations demand hyperparameter-free S-BDT training.

In particular, for S-BDT there are 15 hyperparameters tuned per ε: learning rate η, number of trees T, max tree depth d, adaptive leaf noise ratio r_1, gradient clipping bound g^*, Hessian clipping bound h^*, subsampling rate γ, leaf regularization λ, leaf regularization mode, cyclical feature interaction, ignoring split constraints, random splits from candidates, split refinement, initial score, and initial score clipping bound m^*.

From a privacy view, we propose a different direction than related work which mostly chooses from a previously-determined static set of hyperparameters. We tune our hyperparameter selection for S-BDT based on the hypothesis that the number of trees is the most important hyperparameter and suboptimal hyperparameters, such as subsampling rate γ or max tree depth d, can frequently be corrected by choosing a different number of trees: if each tree gets weaker, we need more trees to reach a strong ensemble. The same goes for the dataset: if the dataset doubles in size, we roughly reach a similar signal-to-noise ratio if we double the number of trees.

Key idea. This hypothesis gives rise to our approach: to achieve hyperparameter-free S-BDT training, we fix all hyperparameters except for the number of trees T and dynamically select the best T based on post-processing without additional leakage, i.e., without impact on ε. We tailor this concept to S-BDT, where we reuse the available differentially private fine-grained information in the form of each decision tree of the ensemble (frequently we have hundreds to thousands of trees per ensemble). The idea is to extract the progress this tree makes for the ensemble from the leaves in each tree. We check whether the leaves of a new tree progress by looking at whether the expected regression error has a clear positive/negative signal or the signal comes close to zero. In the zero-case, we stop the training early. It turns out, that we can approximate the expected regression error with only the information available within a leaf, i.e. the noisy gradient sum. Stopping the training early has the effect that each training uses only as few trees as necessary ensuring the following: First, we can save the ε that would have been spent on the remaining trees. Second, we can dynamically decide how many trees we will need, based on the current training progress. In particular, we do not need to blindly stop the training too early or too late, both of which could lead to suboptimal utility.

Structure. In Section 3.1 we show that the leaves of a new regression tree can be used as a close approximation of the direction of the bias error as part of the MSE metric. In Section 3.2 we show an algorithm that leverages this insight for an early tree stopping.

3.1 Theoretical foundation

The mean squared error (MSE) is a common metric for a regression predictor like a BDT. Given the true label y and predicted label $\tilde{y}$, the MSE is defined as: $\text{MSE}(y,\tilde{y}) = \frac{1}{n}\sum_{i=1}^n (y_i - \tilde{y}_i)^2$. If each error $e_i := y_i - \tilde{y}_i$ is an iid element of the distribution of errors $\mathcal{E}$, we can also define the MSE as: $\text{MSE}(\mathcal{E}) = \text{E}[\mathcal{E}]^2$. The MSE consists of a bias part $\text{E}[\mathcal{E}]$ and a variance part $\text{Var}(\mathcal{E})$ due to the relation $\text{E}[\mathcal{E}]^2 = \text{Var}(\mathcal{E}) + (\text{E}[\mathcal{E}])^2$ for a random variable $\mathcal{E}$. The bias error – a.k.a. underfitting error – is mostly caused by a systematic error in the learning whereas the variance error – a.k.a. overfitting error – is mostly caused by non-deterministic behavior of data which is not perfectly predictable. As the data is frequently non-deterministic, i.e., consists of an unlearnable white noise and a learnable correlated part, a variance error will always remain as the white noise can not be fitted. In contrast, the bias error can vanish if the correlated part is predicted well. Thus, a predictor that minimizes the bias is often also a good predictor with a small MSE.

As a proxy for stopping the tree training as soon as the bias is small, we determine the bias error difference quotient (i.e. a discretized gradient) with a new tree, i.e., for the t-th tree we have $\sum_{i=1}^n e_i^{(t)} - \sum_{i=1}^n e_i^{(t-1)} = 0$ with $e_i^{(t)} := y_i - \tilde{y}_i^{(t)}$ as the error term of the t-th tree on (test) input (x_i, y_i). If the bias error difference quotient reaches zero, we have an extremum (i.e., maximum, minimum, or saddle point) for the bias error. A saddle point is not stable and will likely vanish if we train until the $(t+1)$-th tree. A maximum, as well as a minimum, indicates that the bias that is unsigned has grown/fallen during training and now reaches a saturation point. If we assume that we have a useful configuration and thus make progress during training, reaching the first extremum indicates a good heuristic that the bias might have reached a small value.

We simplify the bias error difference quotient:

$$\begin{aligned}&\textstyle\sum_{i=1}^n e_i^{(t)} - \sum_{i=1}^n e_i^{(t-1)}\\ &= \textstyle\sum_{i=1}^n y_i - \tilde{y}_i^{(t)} - \left(\sum_{i=1}^n y_i - \tilde{y}_i^{(t-1)}\right)\\ &= \textstyle\sum_{i=1}^n \tilde{y}_i^{(t)} - \tilde{y}_i^{(t-1)}\end{aligned}$$

per definition: $\tilde{y}_i^{(t)} = \text{init\_score} + \eta \sum_{t=1}^T \phi^{(t)}(x_i)$ with $\phi^{(t)}(x_i)$ as the t-th tree prediction on input x_i

$$\begin{aligned}&= \eta \textstyle\sum_{i=1}^n \left(\sum_{t=1}^T \phi^{(t)}(x_i) - \sum_{t=1}^T \phi^{(t-1)}(x_i)\right)\\ &= \eta \textstyle\sum_{i=1}^n \phi^{(t)}(x_i).\end{aligned}$$

The term $\phi^{(t)}(x_i)$ depends on the (test) data, yet we can use an approximation that uses the fact that for the training data, we already approximated $\phi^{(t)}(x_i)$ with the leaf values $V^{(t)}(I_{Leaf})$ (cf. Equation (1)) where I_{Leaf} denotes the set of data points x_i that ended up in a leaf:

$$\phi^{(t)}(x_i) \approx V^{(t)}(I_{Leaf}) = -\frac{\sum_{(x_i,y_i)\in I_{\text{Leaf}}} g^{(t)}(\phi^{(t)}(x_i),y_i)}{\sum_{(x_i,y_i)\in I_{\text{Leaf}}} h^{(t)}(\phi^{(t)}(x_i),y_i)+\lambda}.$$

Since for all $x_i \in I_{\text{Leaf}}$ the leaf value is the same, we can rewrite our bias error correction as

$$\begin{aligned}&\eta \textstyle\sum_{i=1}^n \phi^{(t)}(x_i)\\ &\approx \eta \textstyle\sum_{i=1}^n V^{(t)}(I_{Leaf})\\ &= \eta \textstyle\sum_{j=1}^{\#\text{leafs}} \sum_{(x,y)\in I_{\text{Leaf}}} V^{(t)}(I_{Leaf})\end{aligned}$$

since for regression the Hessian $h^{(t)}(\phi^{(t)}(x_i),y_i) = 1$

$$\begin{aligned}&\overbrace{=}^{\text{if } \lambda=0} -\eta \textstyle\sum_{j=1}^{\#\text{leafs}} \sum_{(x_i,y_i)\in I_{\text{Leaf}}} g^{(t)}(\phi^{(t)}(x_i),y_i)\\ &= -\eta \textstyle\sum_{i=1}^n g^{(t)}(\phi^{(t)}(x_i),y_i).\end{aligned}$$

Thus, we can conclude that if we stop the training when the sum of the gradients is zero, the bias error difference quotient is approximately zero, thus we probably most likely reach a saturation point in the bias which may keep the learnable correlated component of the MSE small. Naturally, if the systematic "bias" error vanishes, only the variance error is left which marks a good heuristic for an early tree-stopping point. Moreover, the sum of the gradient within each leaf is already differentially privately released, thus only post-processing is applied without spending extra privacy budget.

Algorithm 1: EarlyStoppedSBDT : Early stop the S-BDT training

```
Input: $\sigma^2_{\text{leaf}}$ : leaf noise scale, $d$ : depth of trees
       $(r_1, r_2)$ : noise weights for leaf value
       $g^*$ : gradient clipping bound
       $T$ : number of rounds; $\sigma_{\text{CI}}$: cutoff CI
1  init$_0$ = DPInitScore (...)
2  $\tau = \sigma_{\text{leaf}} \cdot g^* \cdot \sqrt{1/2r_1} \cdot \sqrt{2^d}$;        // cutoff thr.
3  $\Sigma g = 0$ & dir($\Sigma g$) = UNDEF
4  for $t = 1$ to $T$ do                                  // train tree$_t$
5      if dir($\Sigma g$) = POS then $\Sigma g = \min(\Sigma g, 0)$;
6      else if dir($\Sigma g$) = NEG then $\Sigma g = \max(\Sigma g, 0)$;
7      tree$_t$ = TrainSingleTree(...)
8      $E$ = (init$_0$, tree$_1$, ..., tree$_t$)
9      $\Sigma g$ += sum of noisy gradients in tree$_t$
10     if dir($\Sigma g$) = UNDEF then
11         if $\Sigma g \leq -5 \cdot \tau$ then dir($\Sigma g$) = NEG;
12         if $\Sigma g \geq 5 \cdot \tau$ then dir($\Sigma g$) = POS;
13     $\varepsilon_t$ = RDPAccountant $(\ldots, \sigma^2_{\text{leaf}}, t, \ldots)$
14     $\tau_{\text{CI}} = 10^{\varepsilon_t} \cdot \sigma_{\text{CI}} \cdot \tau$
15     if $t \geq 10 \wedge ((\text{dir}(\Sigma g) = POS \wedge \Sigma g \leq -\tau_{CI}) \vee (\text{dir}(\Sigma g) = NEG \wedge \Sigma g \geq \tau_{CI}))$ then
           break;
16 return E
```

3.2 Algorithm Overview

Algorithm 1 leverages our theoretical insight from Section 3.1 for an early tree stopping heuristic where we halt as soon as the sum of gradients across the leaves of a tree is small. Since the gradients are noisy because of differential privacy, halting exactly at the point where the bias error difference quotient is very small leads to a suboptimal outcome because the quotient can be small just by the influence of the noise.

We modify the classical S-BDT training routine (cf. Algorithm 4) which is indicated in Lines 1, 4, 8 and 16 as follows:
Dynamic cutoff threshold Lines 2, 5, 6, 9 and 13 to 15 describes the dynamic cutoff threshold using the definition of Σg: The cumulative sum of gradients across the trees after a change of direction of a gradient sum. For instance, in the first 20 trees the gradient sum is positive, and starting at tree 21 it is negative. Then, we take the cumulative sum of all gradients starting from tree 21 until the cumulative sum is either smaller or equal to the stopping criteria τ_{CI} or the cumulative sum is larger or equal to zero again which repeats this process. This way we ensure that we only halt if there is a clear change in the sign of the gradient sum which indicates an extremum in the bias error.

The cutoff threshold τ_{CI} is chosen as follows: First, we calculate the expected standard deviation of the noise on the gradient sum Σg. Here, τ depends on the standard deviation applied on each leaf $\sigma_{\text{leaf}} \cdot g^* \cdot \sqrt{1/2r_1}$. As we sum the gradients over 2^d many leaves, we also sum the noise over this many leaves. For $n = 2^d$ normally distributed random variables $X_1, \dots, X_n$ with said standard deviation, we know that the sum of these random variables, i.e. $\sum_{i=1}^{n} X_i$, equals to a random variable with a standard deviation of $\sqrt{\sum_{i=1}^{n} \sigma_{\text{leaf}}^2 \cdot g^{*2} \cdot 1/2r_1} = \sigma_{\text{leaf}} \cdot g^* \cdot \sqrt{1/2r_1} \cdot \sqrt{2^d} =: \tau$.

Based on τ, we further scale the cutoff with a multiplier of σ_{CI} similar to a confidence interval to ensure with high certainty that the current change in direction of the gradient sum signal was due not due to noise. We additionally find that scaling τ with 10^{ε_t} provides a dynamic aspect to the early tree stopping which does not try to accept a constant number of trees among all ε values, but to accept more trees for higher ε values. At higher ε values, the training uses less noise, so we trust the self-correction of the learning process (i.e. use a higher cutoff τ_{CI} that is less sensitive to fluctuations) more than for lower ε where the noise is high such that one noise sample may destroy all learning process. We experimentally observed that the additional dynamic aspect of early stopping is especially useful for Abalone [48] regression dataset, while for the classification datasets adult [6] and spambase [33] we observed neither a clear improvement nor a clear downside.

Dynamic detection of the gradient direction Lines 3, 5, 6, 10 to 12 and 15 describes this part using the definition of $\text{dir}(\Sigma g)$: the sign of how the gradient sum early progresses during training. As the bias error is unsigned, training may progress with a diminishing or growing bias error correction quotient. Thus, we determine in Lines 10 to 12 in the first iterations the sign of the gradient sum to allow to detect and stop after a significant change in sign with threshold τ_{CI}. As the gradient sums are noisy, we use a high $(\sigma = 5)$-confidence interval to determine a positive gradient sum (POS) or a negative one (NEG).

4 TEE Deployment Challenges

Global differential privacy requires a trusted party as a data aggregator, but finding a trusted entity can be difficult in many practical scenarios. Therefore, the use of a hardware-based Trusted Execution Environment (TEE) [59], hosted on an untrusted server, has often been proposed as a practical deployment option for distributed machine learning systems with DP guarantees.

In short, the idea is to send training data to the TEE, which can then run the training algorithm and store the trained model. After training, untrusted parties, such as data analysts, can send inference queries to the model that is only accessible inside the TEE. The TEE can respond to queries while enforcing important restrictions such as the DP algorithm's privacy budget. The intention is that during both the training and query phases, the untrusted server operator, who might collude with data analysts, should not learn anything about the training data or the trained model.

TEE limitations. Despite their attractive security advantages, current TEE technologies also have limitations. Perhaps the main problem with current TEEs is that they share physical computing resources (CPU, caches) with untrusted code running on the same platform, which in practice has led to numerous side-channel attacks demonstrated against all popular TEEs [40]. Side-channel attacks violate the data confidentiality guarantee. In the case of a DP-ML system, this could mean that the untrusted server operator can learn something about the training data or the trained model.

Another limitation is that TEEs do not typically provide any protected non-volatile memory. Once a protected execution is completed inside a TEE and its results need to be stored, persistent storage must be organized outside the TEE. Most TEE technologies support a mechanism called *sealing*, which allows the TEE to encrypt data such that it can be safely passed to the untrusted OS that can store it on disk. Importantly, sealing protects the confidentiality of stored data but not its *freshness*. When the TEE needs to obtain data from persistent storage, the untrusted OS can provide an outdated version of sealed data. In case of DP algorithm, such a *rollback attack* [45, 52] could, for example, trick the TEE to answering more queries than the privacy budget should allow. We note that rollback attacks do not violate either of the TEEs main security guarantees. Instead, they highlight that data confidentiality and execution integrity are insufficient to protect all applications.

This discussion illustrates that simply deploying DP algorithms inside TEEs is often insufficient. Instead, one must carefully address the TEEs' main limitations such as side channels and rollback attacks. Unfortunately, majority of DP research literature that mention TEEs as deployment option focus on algorithmic challenges and improvements, and ignores such deployment aspects, which are, in fact, crucial for DP guarantees.

Prior work and our focus. A recent work called ElephantDP [36] is one of the few research papers to consider TEE deployment challenges of DP algorithms. While we consider this work a great starting point and a solid research contribution, we also observe that it has limitations. For one, the ElephantDP paper focuses on protecting DP guarantees during the query phase, but it does not consider the training of ML algorithms which presents another significant attack surface. The second limitation is that this work does not consider side-channels, which are perhaps the main security problem of current TEEs. Moreover, our analysis shows that the rollback protection mechanism proposed in ElephantDP creates an unrestricted oracle for side-channel attacks, thus enlarging the above two problems.

In the following two sections, we consider the safe deployment of TEE-based DP algorithms more broadly. This includes addressing security challenges that arise both during the training and deployment phases of ML system. It also involves designing and implementing protections that address side-channel leakages and rollback attacks, the two main security issues of current TEEs. In particular, we show that the DP algorithm presented in this paper can be implemented using constant-time programming practices to address side-channel leakages (both during the training and query phase). We also outline an improved rollback protection approach that follows a prior work called Memoir [52] and additionally turns randomized DP algorithms into deterministic executions to prevent side channel leakage.

5 Implementing S-BDT on Trusted Hardware

Vanilla S-BDT [53] is unprotected from leakage through timing side-channels and can potentially expose sensitive attributes of its training data to an untrusted entity that is able to track runtime behavior of S-BDT's execution (e.g. execution time, branching behavior). A simple example how a DP attacker can exploit this leakage is displayed in Figure 1.

In the following sections we identify the reasons for these leakages and describe how we fix them in our constant-time adaptation of S-BDT. The complete pseudocode for constant-time S-BDT is in Appendix B. In the experimental section (cf. Section 7.3) we validate our constant-time implementation using Microwalk [65]. We port our implementation of constant-time S-BDT with early tree stopping (cf. Section 3) to an Intel TDX virtual machine [34] (cf. Section 5.1).

Data-dependent memory access A program that has data-dependent memory access patterns may leak the patterns and thus parts of the data through a cache that is shared with an untrusted entity [9,21,30,31,47,60]. When DP approximating a leaf value (cf. Algorithm 7 in Line 1 and Line 3), vanilla S-BDT only sums up those data points that are in a specific leaf, allowing the DP adversary to infer sensitive attributes based on the specific leaf a data point ends up in. In Algorithm 4 vanilla S-BDT only updates the individual privacy budgets of data points that are not filtered out yet, leaking the specific data points that are still active. And in Algorithm 11 vanilla S-BDT computes gradients and Hessians only for the data points of the current subsample, effectively removing privacy amplification by subsampling because the subsample can be exposed to the DP adversary. In our constant-time version, we always iterate over all data points, filtering them with bitmasks. We use one bitmask indicating whether a data point is in the subsample of the current training round, one bitmask indicating data points that are still active after individual privacy filtering, and one bitmask for each leaf indicating the data points that end up in that leaf.

Data-dependent control flow Similar issues to data-dependent memory access arise when the control flow of a program is data-dependent. Powerful attacks like single-stepping allow a malicious hypervisor to precisely track the control flow of a TEE [11,67] and thus infer sensitive data.

When computing the prediction of a data point on a tree, vanilla S-BDT directly evaluates only the path leading from the tree root to the leaf that the data point ends up in. This control flow can leak the leaf, revealing sensitive attributes of that data point. In our fixed version of Algorithm 10, the entire tree always gets traversed in the same order, effectively removing data-dependent control flow. The correct leaf value is then returned via a constant-time selection based on the sensitive attribute, replacing branching on the sensitive attribute which is not constant-time. The selection algorithm is displayed in Algorithm 2 and can be made constant-time by replacing all arithmetic operations with their equivalent operations in the libftfp library [3]. In Algorithm 7 the indicator functions are also replaced with this constant-time selection.

Algorithm 2: Constant-time selection

```
  Input: $b$: boolean value, $u,v$ two values
1 return $b \cdot u + (1-b) \cdot v$
```

Data-dependent runtime of instructions Instructions like addition and multiplication can leak the value of sensitive data. For example, multiplying two numbers on a Core i7 processor will take 4 clock cycles, but when the operands are subnormal, i.e., hold values close to zero, the same multiplication can take over 200 clock cycles [3], a difference in runtime that gets further amplified by repeated computation inside of a loop. A DP adversary who carefully chooses their input can thus leak the value of sensitive data through the runtime of such instructions. To counter this problem, we replace all operations that use sensitive data, i.e., computing the individual privacy budget $\rho_t^{(i)}$ (cf. Algorithm 4), computing the sum over labels and adding the noise in the DP initial score (cf. Algorithm 5), computing the sums over gradients and Hessians and adding the noise when DP approximating a leaf value (cf. Algorithm 7), computing the prediction of the ensemble (cf. Algorithm 9 and Algorithm 10) and computing gradients and Hessians (cf. Algorithm 11), with their equivalent instructions from the libftfp library [3], which all run in constant-time and independent of the operand's values.

Leaking noise samples Leaking noise samples that are sampled from the Gaussians in the DP initial score (cf. Algorithm 5) and for DP approximating a leaf value (cf. Algorithm 7) breaks DP because an adversary observing the samples can use them to denoise the DP approximated releases, e.g., in the DP initial score (cf. Line 9) if the adversary knows the noise from Gaussian was $s = 0.123$ and the DP approximated value of $M_{\text{priv}} = 1.123$, then the nonprivate value must be $M = 1.0$. Typical Gaussian samplers are not constant-time and thus suffer from side-channel vulnerabilities. We utilize FACCT [69], a constant-time implementation

Algorithm 3: Constant-time S-BDT with randomized input length

```
    Input: $D$: data set
1   $\tilde{l} = \text{input\_len} + \mathcal{N}(0, 1/\varepsilon_{\text{input}})$
2   Run constant-time S-BDT with input length $\tilde{l}$
3   Write back resulting ensemble to output.
```

of a discrete Gaussian D_σ. We set $\sigma = 2048$ to be large so that when scaling the resulting discrete distribution by σ, i.e. D_σ/σ we obtain a reasonable approximation of a continuous standard normal $\mathcal{N}(0,1)$.

S-BDT uses a Bernoulli distribution to generate a Poisson subsample (cf. Algorithm 12) for privacy amplification. Typical Bernoulli samplers are not constant-time and potentially leak what samples end up in a subsample, breaking the privacy amplification because the subsample is then known to the adversary. We replace the Bernoulli distribution with the constant-time version that is a subroutine of FACCT [69].

Leaking input length In unbounded DP, where the adversary distinguishes between two different inputs that differ in their length by one element, exposing the length of the input breaks DP guarantees. Similarly, vanilla S-BDT traverses the input multiple times during its execution, opening the possibility for leaking the input length. We remedy this problem by first releasing the input length l via the Gaussian mechanism $\tilde{l} := l + \mathcal{N}(0, 1/\varepsilon_{\text{input}})$ and then truncating the input or padding it with placeholder data to have the length match the DP approximated $\tilde{l}$ (cf. Algorithm 3). In our constant-time version, S-BDT traverses the entire data set in the loop of Algorithm 4, or in the sums in Algorithm 5 and Algorithm 7 so the adversary can only learn $\tilde{l}$ from observing the runtime, which is fine for DP.

In practice, we assume that the initial padding or truncation of data has been performed by the data owner and prior to running S-BDT, so S-BDT will only leak the DP approximated input length when reading the input file.

Timing Privacy We prove Timing Privacy [56] in an idealized RAM model to validate our construction of Algorithm 3. Timing Privacy demands the randomized running time of a mechanism not to leak much more than what is already revealed by the output distribution (cf. Section 2.3).

We assume that all hyperparameters are baked into Algorithm 3 and are not part of the input. We note that the DP approximated input length $\tilde{l}$ (cf. Algorithm 3) can be reused instead of being recomputed for the DP initial score (cf. Algorithm 5). We use the ω-bit Word RAM model's oracle `RAND` to sample uniform noise. We extend the model by oracles for selection between two values u, v based on a boolean selector b (cf. Algorithm 2), logarithm, binomial coefficient, exponentiation, Bernoulli sampling and Gaussian sampling. We assume all oracles to perform in a single time step, but note that this assumption is met in practice by making all oracles constant-time using the libftfp [3] and FACCT [69] libraries. We also assume that the sequence of memory cells is large enough to store $\tilde{l}$ data points and that reading from uninitialized memory is safe so we can read placeholder data points from uninitialized memory after the actual data points. We assume that one operation (e.g., calling an oracle or performing a multiplication) can be done per time step. Under these assumptions, we show the following result.

Theorem 9. *Algorithm 3 satisfies* $(1 \mapsto \rho_{\mathit{input}}(\alpha))$*-Timing Privacy.*

Proof. Let $X \sim X'$ be two neighboring data sets and $y \in \mathcal{R}$ an arbitrary output. Let $T_{\max}$ be the number of trees, d the depth of trees, $\alpha_{\max} \cdot \sigma_{\max}$ the number of iterations performed in Algorithm 8. Let $\tilde{l} := |X| + \mathcal{N}(0, 1/\varepsilon_{\text{input}})$ be the DP approximated input length of input X.

By direct analysis (cf. Appendix C for details) we get a constant amount of operations independent of input length

$$K = 2 \cdot \left(4 + 8\alpha_{\max}\sigma_{\max} - T_{\max} + 5 \cdot 2^d T_{\max}\right)$$

and a constant amount of iterations for each data point in the input

$$k = 2 + \frac{(35 + 9 \cdot 2^d + 22\alpha_{\max})T_{\max}}{2} + \frac{(1 + 3 \cdot 2^d)T_{\max}^2}{2}$$

Let M be the mechanism of Algorithm 3. We compute the Timing Privacy guarantee of M:

$$\begin{aligned}
&D_\alpha(T_M(X, \text{env})|_{\text{out}(M(X,\text{env}))=y} \,||\\
&T_M(X', \text{env}')|_{\text{out}(M(X,\text{env}))=y})\\
&\quad = D_\alpha(\mathcal{N}(|X| + K, k^2 \cdot 1/\varepsilon_{\text{input}}) \,||\\
&\mathcal{N}(|X'| + K, k^2 \cdot 1/\varepsilon_{\text{input}}))\\
&\quad = D_\alpha(\mathcal{N}(0, k^2 \cdot 1/\varepsilon_{\text{input}}) || \mathcal{N}(1, k^2 \cdot 1/\varepsilon_{\text{input}}))
\end{aligned}$$

Using [46, Proposition 7]

$$\le \alpha/(2 \cdot k^2 \cdot 1/\varepsilon_{\text{input}}) =: \rho_{\text{input}}(\alpha)$$

□

5.1 Porting MammothDP to Intel TDX

We port our implementation of constant-time S-BDT with early tree stopping (cf. Section 3) to an Intel TDX virtual machine [34]. We run it on a TDX-enabled Intel Xeon Gold 6526Y Emerald Rapids processor. A Ubuntu 24.10 image that contains a compiled binary of MammothDP is available at https://drive.google.com/1TW8-TDX-image.

6 Rollback Protection

The execution in modern TEEs leverages volatile memory locations such as registers and cache inside the CPU package. Once the execution is complete, such memory locations will be erased to ensure data confidentiality. Any persistent storage needed must be organized outside the TEE. To support this, most TEE architectures support a storage mechanism called *sealing* [20]. This allows the TEE application to encrypt data using authenticated encryption, such that only the same TEE application that sealed can later decrypt (unseal) it and verify its authenticity. In the context of a differentially-private ML system, the TEE could, for example, seal the computed model after its training. The TEE application should also update the privacy budget for every answered query and store the latest budget value to disk using sealing.

The sealing mechanism provided by current TEEs is susceptible *rollbacks* (sometimes also called *rewinds*) [45, 52]. When the TEE application needs to load its latest state (e.g., after being restarted by the OS) it can ask the OS to serve the latest sealed state from disk. Instead of serving the latest sealed data, the OS can provide a previous version of sealed data to the TEE. Towards the TEE application everything looks correct, as the sealing is cryptographically correct (authenticated using a valid key), but the rollback attack effectively rewinds the TEE's state to a previous privacy budget value and thus enables violation of DP guarantees.

Rollback protection requirements. To address rollback attacks, some form of trusted, non-volatile memory is needed. In research literature such trusted memory that resides outside the TEE is typically called State Continuity Module (SCM) [52]. One possible instantiation is a trusted hardware element such as a TPM chip which is equipped with non-volatile DRAM. The main downside of hardware-based SCM is that non-volatile memory writes are typically slow and the NVRAM memory will wear out with each write [45,52]. This makes applications that require very frequent SCM updates infeasible using hardware SCMs like the TPM chip.

Another option is to leverage a distributed state-keeping system as the SCM. For example, several research papers have explored the idea of a distributed system created by multiple SGX enclaves [4,45,49]. Such SCM instantiations do not have similar update frequency restrictions, but they do increase the overall system complexity and introduce additional trust assumptions (e.g., the setup phase typically requires a trusted entity that defines which nodes to enroll to the distributed system).

A good rollback protection system should support two properties. The first is *safety*, which means that the TEE application should only load the latest version of its sealed data. The second is *liveness*, which means that even if the TEE suddenly crashes, or the underlying computing platform loses power, the TEE can continue its operation after a restart.

Prior work. A recent work called ElephantDP [36] proposes a rollback protection mechanism for differentially-private ML systems. The goal of this solution is to ensure that a privacy budget managed by the TEE is not rolled back to a previous value.

ElephantDP achieves this by first executing the DP algorithm and processing the inference query that is received from an untrusted data analyst, then saving the TEE application's state (including the updated privacy budget and the *response* to the query) on disk, and finally updating the SCM with the hash of the TEE application's state. Assume the TEE crashes after updating the SCM, but potentially before being able to send the response to the analyst. The TEE application can be restarted and the system ensures that only the same response will be outputted. The paper argues that such TEE restarts and re-executions do not provide any advantage to the DP adversary.

However, this approach provides an advantage to a side-channel adversary, since it creates an unrestricted execution *oracle* that can be used to launch side-channel attacks. To illustrate this, let us consider the following example. The untrusted data analyst sends a query to the TEE application. The colluding OS aborts the execution of the TEE after it has processed the query (i.e., executed the DP algorithm with the adversary-chosen input), but before the local storage or the SCM is updated. Then these two steps can be repeated with a new query value. This means that ElephantDP prevents the enclave from returning a different output, but it will not prevent the TEE application re-execution itself. This allows a side-channel adversary, such as the untrusted OS colluding with the data analyst, to execute the TEE unlimited number of times with freely chosen inputs.

Side-channel leakage can be addressed using defensive programming techniques, as we have done in this paper. However, ensuring that the compiled code exhibits constant-time properties is challenging. Even if one executable is verified to behave in constant-time manner, a different compiler, compiler settings, or computing architecture may introduce subtle changes to the executable and violate constant-time properties [58]. In addition, writing code following the constant-time paradigm can be tedious and slow. Not every real-life project may afford such time investment. Ideally, the rollback protection mechanism should be free of such side-channel oracles, if it can be done without sacrificing safety or liveness.

Eliminating side channel oracle beyond timing-leakage. Next, we outline that this is possible by using a previous work called Memoir [52] as a starting point. For each received query, the TEE should *first update the SCM* with a hash of its current state. The TEE application's state should include a history of all previous inputs (queries). Once the SCM is updated, the TEE can execute the DP algorithm with the received query input and provide an output response to the data analyst. If the TEE crashes after the SCM is updated, but potentially before the response is sent to the analyst, the TEE can be re-executed using the same input and its previous state. During such re-execution the TEE can verify, by consulting the SCM, that the same input is provided or abort its execution otherwise.

This basic approach, proposed in the Memoir paper, ensures that such TEE re-executions after a crash are only possible with the same input. Assuming that the executed algorithm is deterministic, the side-channel adversary learns nothing new from the repeated execution. However, DP algorithms are randomized, and therefore this approach alone is not sufficient to eliminate the side-channel oracle. The adversary (untrusted OS) could re-execute the randomized DP algorithm with the same input many times and observe a different side-channel trace for each execution.

Fortunately, this remaining problem can be solve easily in the context of DP ML systems. The data owners, who provision training data to the TEE, can each provide a randomness seed before sending any training data. The TEE application can combine such received seeds on its sealed state. The TEE application can be configured to only accept training data from a data owner if it has already contributed a randomness seed that is included to TEE application's state. During training and query processing, the TEE application can then create the needed pseudo-randomness using PRNG and the combined seed (e.g., all received seeds are concatenated and hashed together). This approach will make the TEE's execution deterministic and eliminate the side-channel oracle from the rollback protection mechanism.

This discussion illustrates that the two well-known limitations of modern TEEs, side-channels and rollback attacks, are actually connected problems. And careful system deployment consideration is needed to address both problems simultaneously.

7 Evaluation

Sensitive Datasets. We use the three datasets that were used in the S-BDT evaluation [53] for our experiments: Abalone, Adult and Spambase. Abalone [48] is a regression dataset containing 4,177 data points. Given eight numerical attributes, e.g. sex, length, and diameter of an abalone, the task is to predict its age. Adult [6] is a binary classification dataset with more than 48,000 data points. Given 14 attributes, e.g. age, sex, and occupation, the task is to determine whether a person earns over 50,000 $ per year. Spambase [33] is a binary classification dataset containing 4,601 data points. Given 57 numerical attributes, e.g. frequencies of words and symbols in a mail, the task is to determine whether that mail is spam.
Experimental Setup. We set $\delta = 1.5 \cdot 10^{-7}$ for Abalone and $\delta = 3 \cdot 10^{-8}$ for Adult and Spambase. Tests were run with an Intel Xeon Platinum 8168 2.7 GHz CPU and 32GB of RAM. The code of MammothDP is available at https://anonymous.4open.sciencer/mammothdp.

7.1 Early Stopping

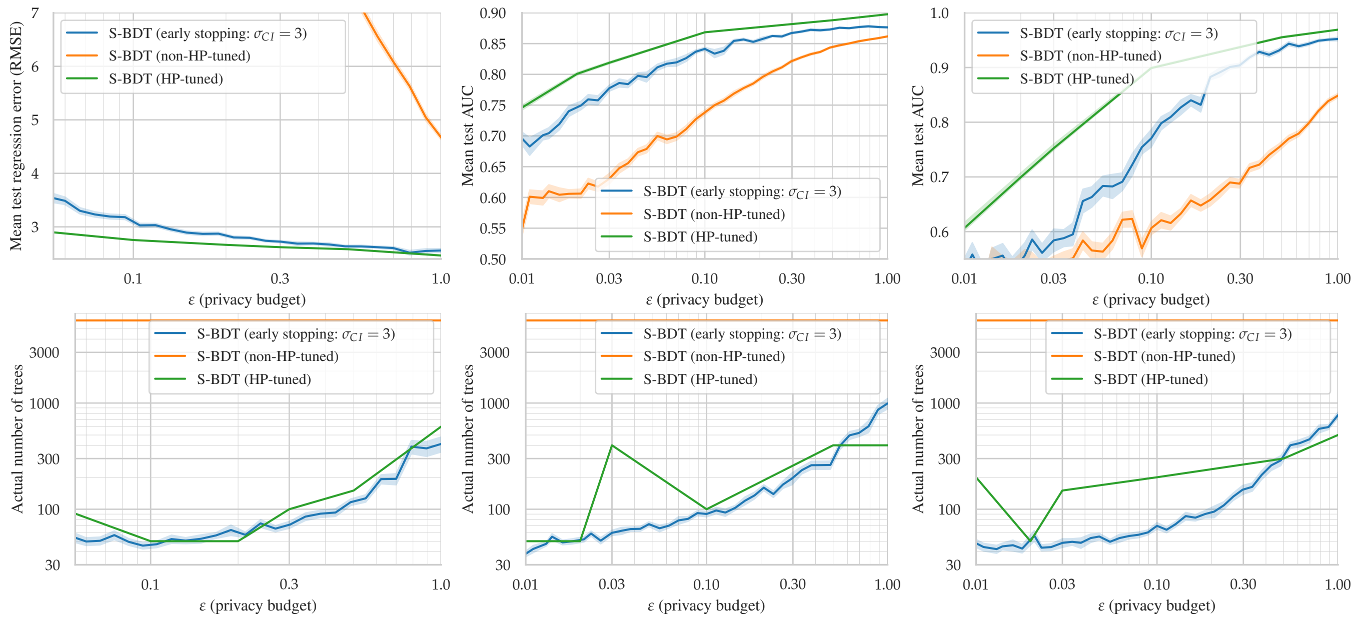
In Figure 2, we compare S-BDT with hyperparameter-free early stopping (cf. Algorithm 1 on $\sigma_{\text{CI}} = 3$) to hyperparameter-free S-BDT and as a reference, we include the privacy-leaky hyperparameter tuned S-BDT [53, Figure 3].

We observe that hyperparameter-free early stopped S-BDT significantly outperforms hyperparameter-free S-BDT on all three datasets. We reach a less but comparable utility when comparing early stopped S-BDT to the reference hyperparameter-tuned S-BDT. In the Appendix, we show in Figure 4 that this gap closes the smaller we set the maximal number of allowed trees. Moreover, Figure 2 also shows that early stopped S-BDT mostly selects a similar number of trees to the HP-tuned variant which validates our early stopping heuristic.
Static Hyperparameters. For both hyperparameter-free variants, we have to choose a static set of reasonable hyperparameters. We used a compromise of the best-performing hyperparameters in HP-tuned S-BDT on abalone, adult, and spambase. For deployment, we recommend using publically available datasets. The chosen parameters are the following: learning rate $\eta = 0.1$, maximal number of trees $T = 6{,}000$, max tree depth $d = 2$, adaptive leaf noise ratio $r_1 = 0.4$, gradient clipping bound $g^* = 0.2$, Hessian clipping bound $h^* = 0.2$, subsampling rate $\gamma = 0.2$, leaf regularization $\lambda = 15$, leaf regularization mode `ADD`, enabled cyclical feature interaction, enabled ignoring the split constraints, enabled random splits from candidates, disabled split refinement, activated initial score solely for regression datasets, and initial score clipping bound $m^* = 0.5$.

Additionally, we performed some feature pre-processing like rescaling some attributes and categorizing the attributes into categorical or numerical ones based on the meta-data description of a dataset. For comparability, we used the same numerical feature range as S-BDT: for abalone we have a range of $0 \ldots 0.5$, for adult of $0 \ldots 100$, and for spambase of $0 \ldots 1$. We recommend a data-independent numerical feature range for deployment.

7.2 Privacy - Utility tradeoffs

We compare vanilla S-BDT with our constant-time S-BDT using the best perfoming sets of hyperparameters reported for vanilla S-BDT [53]. For the non-private baselines we report performance of the XGBoost [17] algorithm implemented in the S-BDT framework and denote *xgboost* for reporting the utility when selecting the optimal, data-dependent split and *xgboost random splits* when selecting splits uniformly at random from the split value range. Our findings are displayed in Figure 3. On Spambase and Adult, vanilla and constant-time S-BDT perform equally. On Abalone we observe a very small decrease in RMSE for constant-time S-BDT. This is



(a) Abalone (b) Adult (c) Spambase

Figure 2: S-BDT with privacy-leaky hyperparameter-tuning (green), with statically chosen hyperparameters (orange), or with early stopping (blue). We plot the privacy-utility tradeoff (top) and the selected number of trees (bottom). Evaluated with 10-repeated 5-fold cross-validation and an upper bound of 6,000 trees.

Dataset	Setting	Time cost
Abalone	Vanilla S-BDT	1.9 s
Abalone	Constant Time S-BDT	48.0 s
Adult	Vanilla S-BDT	3.0 s
Adult	Constant Time S-BDT	58.4 s
Spambase	Vanilla S-BDT	2.8 s
Spambase	Constant Time S-BDT	45.3 s

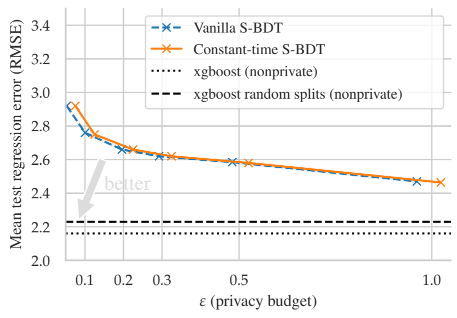
Table 1: **Time cost** of Vanilla S-BDT and Constant Time S-BDT. We train $T = 100$ rounds with tree depth $d = 6$.

due to replacing the Laplace mechanism in the DP initial score from vanilla S-BDT with the Gaussian mechanism (cf. Algorithm 5): While the Laplace mechanism is $(\varepsilon, 0)$-DP, the Gaussian mechanism is (ε, δ)-DP. We use the same δ that we use for the Gaussian mechanism in DP approximating the leaf (cf. Algorithm 7) for the two invocations of the Gaussian mechanism in the DP initial score (cf. Algorithm 5) which increases the overall δ from $\delta = 3 \cdot 10^{-8}$ to $\delta = 9 \cdot 10^{-8}$. We also increase the privacy budget ε_{ds} used in the DP initial score (cf. Algorithm 5) from $\varepsilon_{\text{ds}} = 0.005$ to $\varepsilon_{\text{ds}} = 0.025$.

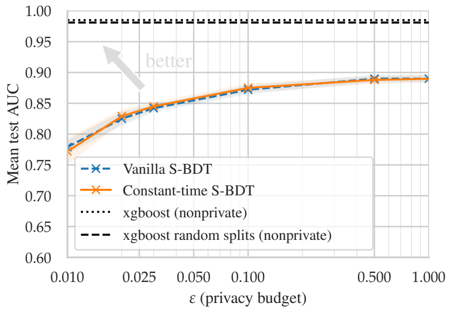
7.3 Constant-Time Evaluation

To verify that our compiled implementation is indeed constant-time, we analyzed it with Microwalk. For that, we divided data-dependent parts of our implementation into the independent components DP Initial Score, computing and updating the individual privacy budgets, computing gradients and Hessians, forming a leaf node and adding Gaussian noise with fixed randomness, generating the bitmask that notes which data point ends up in which leaf of a tree and prediction of the ensemble. For each component, we generated 100 random test inputs. These test inputs are loaded by thin wrapper functions, which transform the test inputs into the correct format (while maintaining uniformity) and then call the target component. We embed the wrapper functions into Microwalk's new analysis template [26], which automates communication with Microwalk and loading and executing the test inputs. The resulting analysis workflow is fully automatic, which makes re-verifying the full implementation after changes straightforward.

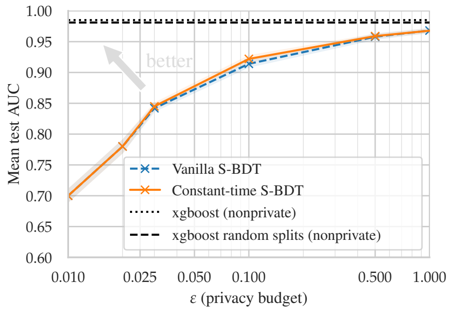
Microwalk successfully verifies all components, without reporting any leakages. The (single-threaded) analysis of all components takes 76 seconds in total (average 7.6 seconds per component), at a memory consumption of at most 300 MB. We verified that the analysis covered all executed S-BDT and libftfp code. Increasing the number of test cases does not lead to higher coverage. We conclude that our S-BDT implementation exhibits no data-dependent



(a) Dataset: **Abalone** (Regression)



(b) Dataset: **Adult** (Classification)



(c) Dataset: **Spambase** (Classification)

Figure 3: Comparison of utility-privacy tradeoff of vanilla S-BDT and constant-time S-BDT. Regression error (RMSE) (Abalone) and AUC (Adult and Spambase) vs. privacy budget ε ((b) and (c) in log-scale). We set $\delta = 1.5 \cdot 10^{-7}$ for Abalone and $\delta = 3 \cdot 10^{-8}$ for Adult and Spambase. Evaluated with 20-repeated 5-fold cross-validation. The transparent area is the standard error.

control flow or memory accesses, making it robust against system-level attackers. Code of our evaluation is available at https://anonymous.4open.science/r/mwsbdt.

Runtime overhead We investigate the time cost of vanilla S-BDT and our constant-time S-BDT. The constant-time implementation increases runtime due to the fixed-time instructions taking more clock cycles to run [3, Figure 14] and due to all loops in the constant-time S-BDT always traversing the entire data set instead of only a much smaller subsample or a much smaller collection of data points inside a specific leaf. We observe an overhead of factor 15 to 25 (cf. Table 1).

8 Discussion & Limitations

The early stopping method requires a sufficiently large upper bound for the number of trees (we chose 6,000) to ensure that the tree learner has enough expressivity for the learning task. However, in some cases, randomness leads the early stopping astray and no stopping occurs, leading to weak performance and high privacy leakage. A comparison of the Spambase (Figure 3c) and Abalone (Figure 3a) which are of similar dataset size results show that not all data sets work equally well. To understand, further research into this early stopping method is needed.

The constant-time implementation has a well-known significant efficiency overhead, which in some applications might be prohibitively large. As S-BDT is very efficient, this efficiency-overhead was not a problem in our experiments, though. In practice, it might be too costly to devise a constant-time version of existing methods and harden the TEE.

9 Related Work

The related work on roll-back attacks on TEEs is described in Section 4.

Hardening ML algorithms for trusted hardware Prior works harden various machine learning algorithms against timing side-channels for trusted hardware. Law et al. [38] harden the XGBoost [17] algorithm, Ohrimenko et al. [50] propose constant-time implementations for various ML algorithms, including decision trees or support vector machines.
Hyperparameter-free DP-ML Current work on hyperparameter-free differentially private machine learning algorithms mainly focuses on finding the best hyperparameter from a previously specified set of hyperparameter choices. In that, there are three approaches for hyperparameter-freeness: either via heuristics [1, 35], via separating the data in disjoint sets [15, 37], or via spending extra privacy [1, 16, 43, 51]. Spending extra privacy budget means more noise is used in training when the overall privacy budget is fixed which deteriorates utility. Dividing the data set into disjoint sets implies using less data in training which also deteriorates utility. In this work, we go a step beyond heuristic-based solutions by adaptively tuning the important hyperparameter of the number of trees during training using only the differentially private output of the model.

10 Conclusion

We introduced a novel hyperparameter-freeness method, via early stopping, that does not incur any privacy-overhead. Towards a practical deployment of ML algorithms in EHDS, we have introduced a TEE-hardened practical DP tree ensemble learner for which the side-channel analysis tool Microwalk did not find side-channel leakage. Our experiments show that despite the efficiency overhead of the constant-time implementation, the running time is acceptable. Our experiments further show that the hyperparameter-freeness method outperforms non-hyperparameter-tuned DP tree ensemble learners, and is comparable to hyperparameter-tuned DP tree ensemble learners, for which the hyperparameter-tuning is non-DP.

11 Ethics considerations

This work solely works on benchmark data; so, we do not see any cause for ethical concerns.

12 Compliance with the open science policy

In this work we share the source code of our implementation, the source code of our evaluation and a compiled binary inside an Intel TDX virtual machine.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 308–318, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. On the power of simple branch prediction analysis. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*, ASIACCS '07, page 312–320, New York, NY, USA, 2007. Association for Computing Machinery.

[3] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. On subnormal floating point and abnormal timing. In *2015 IEEE Symposium on Security and Privacy*, pages 623–639, 2015.

[4] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. Nimble: rollback protection for confidential cloud services. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 193–208, 2023.

[5] Borja Balle, Gilles Barthe, Marco Gaboardi, Justin Hsu, and Tetsuya Sato. Hypothesis testing interpretations and renyi differential privacy. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2496–2506. PMLR, 2020.

[6] Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996.

[7] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, and Yann LeCun. Differentially- and non-differentially-private random decision trees. *preprint arXiv:1410.6973*, 2014.

[8] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy*, pages 553–570, 2015.

[9] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: Sgx cache attacks are practical. In *Proceedings of the 11th USENIX Conference on Offensive Technologies*, WOOT'17, page 11, USA, 2017. USENIX Association.

[10] Ernie Brickell, Gary Graunke, and Jean-Pierre Seifert. Mitigating cache-timing based side-channels in aes and rsa software implementations. In *Conference 2006 session DEV-203*. RSA, 2006.

[11] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 178–195. ACM, 2018.

[12] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without page faults: Stealthy page Table-Based attacks on enclaved execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1041–1056, Vancouver, BC, August 2017. USENIX Association.

[13] Clément L Canonne, Gautam Kamath, and Thomas Steinke. The discrete gaussian for differential privacy. In *Advances in Neural Information Processing Systems*, volume 33, pages 15676–15688. Curran Associates, Inc., 2020.

[14] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914, 2022.

[15] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *J. Mach. Learn. Res.*, 12:1069–1109, 2011.

[16] Kamalika Chaudhuri and Staal A Vinterbo. A stability-based validation procedure for differentially private machine learning. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[17] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794. Association for Computing Machinery, 2016.

[18] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel tdx demystified: A top-down approach. *ACM Computing Surveys*, 56(9):1–33, 2024.

[19] European Commission. Proposal for a regulation of the european parliament and of the council on the european health data space, com(2022) 197 final, 2022.

[20] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Paper 2016/086, 2016.

[21] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. Cachequote: Efficiently recovering long-term secrets of sgx epid via cache attacks. *Transactions on Cryptographic Hardware and Embedded Systems*, 2018, Issue 2:171–191, 2018.

[22] Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk. Binsec/rel: Symbolic binary analyzer for security with applications to constant-time and secret-erasure. 26(2):11:1–11:42.

[23] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. Cryptology ePrint Archive, Paper 2013/383, 2013.

[24] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, Berlin, Heidelberg, 2006.

[25] Cynthia Dwork, Moni Naor, Omer Reingold, Guy N. Rothblum, and Salil P. Vadhan. On the complexity of differentially private data release: efficient algorithms and hardness results. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, (STOC)*, pages 381–390. ACM, 2009.

[26] Iliana Fayolle, Jan Wichelmann, Anja Köhl, Walter Rudametkin, Thomas Eisenbarth, and Clémentine Maurice. Semi-automated and easily interpretable side-channel analysis for modern javascript. In *Cryptology and Network Security - 23rd International Conference, CANS 2024, Cambridge, UK, September 24-27, 2024, Proceedings, Part II*, volume 14906 of *Lecture Notes in Computer Science*, pages 25–46. Springer, 2024.

[27] Vitaly Feldman and Tijana Zrnic. Individual privacy accounting via a rényi filter. In *Advances in Neural Information Processing Systems*, volume 34, pages 28080–28091. Curran Associates, Inc., 2021.

[28] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[29] Antoine Geimer, Mathéo Vergnolle, Frédéric Recoules, Lesly-Ann Daniel, Sébastien Bardin, and Clémentine Maurice. A systematic evaluation of automated tools for side-channel vulnerabilities detection in cryptographic libraries. In *2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1690–1704. ACM.

[30] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, EuroSec'17, New York, NY, USA, 2017. Association for Computing Machinery.

[31] Marcus Hähnel, Weidong Cui, and Marcus Peinado. High-Resolution side channels for untrusted operating systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 299–312, Santa Clara, CA, July 2017. USENIX Association.

[32] Mike Hamburg. Accelerating aes with vector permute instructions. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 18–32. Springer Berlin Heidelberg, 2009.

[33] Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermondt. Spambase. UCI Machine Learning Repository, 1999.

[34] Intel. Intel tdx. White Paper, 2023. [Accessed: 2025-01-02].

[35] Roger Iyengar, Joseph P. Near, Dawn Song, Om Thakkar, Abhradeep Thakurta, and Lun Wang. Towards practical differentially private convex optimization. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 299–316, 2019.

[36] Jiankai Jin, Chitchanok Chuengsatiansup, Toby Murray, Benjamin I. P. Rubinstein, Yuval Yarom, and Olga Ohrimenko. Elephants do not forget: Differential privacy with state continuity for privacy budget. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 1909–1923, New York, NY, USA, 2024. Association for Computing Machinery.

[37] Antti Koskela and Tejas Kulkarni. Practical differentially private hyperparameter tuning with subsampling. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[38] Andrew Law, Chester Leung, Rishabh Poddar, Raluca Ada Popa, Chenyu Shi, Octavian Sima, Chaofan Yu, Xingmeng Zhang, and Wenting Zheng. Secure collaborative training and inference for xgboost. In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, PPMLP'20, page 21–26, New York, NY, USA, 2020. Association for Computing Machinery.

[39] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 557–574, Vancouver, BC, August 2017. USENIX Association.

[40] Mengyuan Li, Yuheng Yang, Guoxing Chen, Mengjia Yan, and Yinqian Zhang. Sok: Understanding design choices and pitfalls of trusted execution environments. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 1600–1616, 2024.

[41] Qinbin Li, Zhaomin Wu, Zeyi Wen, and Bingsheng He. Privacy-preserving gradient boosting decision trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):784–791, 2020.

[42] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. Design and verification of the arm confidential compute architecture. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 465–484, 2022.

[43] Jingcheng Liu and Kunal Talwar. Private selection from private candidates. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing(STOC)*, pages 298–309. ACM, 2019.

[44] Samuel Maddock, Graham Cormode, Tianhao Wang, Carsten Maple, and Somesh Jha. Federated boosted decision trees with differential privacy. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2022.

[45] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. {ROTE}: Rollback protection for trusted execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1289–1306, 2017.

[46] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275, 2017.

[47] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How sgx amplifies the power of cache attacks. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 69–90, Cham, 2017. Springer International Publishing.

[48] Warwick Nash, Tracy Sellers, Simon Talbot, Andrew Cawthorn, and Wes Ford. Abalone. UCI Machine Learning Repository, 1995.

[49] Jianyu Niu, Wei Peng, Xiaokuan Zhang, and Yinqian Zhang. Narrator: Secure and practical state continuity for trusted execution in the cloud. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2385–2399, 2022.

[50] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious Multi-Party machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, Austin, TX, August 2016. USENIX Association.

[51] Nicolas Papernot and Thomas Steinke. Hyperparameter tuning with renyi differential privacy. In *International Conference on Learning Representations*, 2022.

[52] Bryan Parno, Jay Lorch, John (JD) Douceur, James Mickens, and Jonathan M. McCune. Memoir: Practical state continuity for protected modules. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2011.

[53] Thorsten Peinemann, Moritz Kirschte, Joshua Stock, Carlos Cotrini, and Esfandiar Mohammadi. S-bdt: Distributed differentially private boosted decision trees. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 288–302, New York, NY, USA, 2024. Association for Computing Machinery.

[54] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM computing surveys (CSUR)*, 51(6):1–36, 2019.

[55] Raluca Ada Popa. Confidential computing or cryptographic computing? *Communications of the ACM*, 67(12):44–51, 2024.

[56] Zachary Ratliff and Salil Vadhan. A framework for differential privacy against timing attacks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page

3615–3629, New York, NY, USA, 2024. Association for Computing Machinery.

[57] Alfréd Rényi. On measures of entropy and information. In *Proceedings of the fourth Berkeley symposium on mathematical statistics and probability, volume 1: contributions to the theory of statistics*, volume 4, pages 547–562. University of California Press, 1961.

[58] Moritz Schneider, Daniele Lain, Ivan Puddu, Nicolas Dutly, and Srdjan Capkun. Breaking bad: How compilers break constant-time˜ implementations. *arXiv preprint arXiv:2410.13489*, 2024.

[59] Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srdjan Capkun, and Ronald Perez. Sok: Hardware-supported trusted execution environments. *arXiv preprint arXiv:2205.12742*, 2022.

[60] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: Using sgx to conceal cache attacks. In Michalis Polychronakis and Michael Meier, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24, Cham, 2017. Springer International Publishing.

[61] Florian Sieck, Sebastian Berndt, Jan Wichelmann, and Thomas Eisenbarth. Util::lookup: Exploiting key decoding in cryptographic libraries. In *2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2456–2473. ACM.

[62] Laurent Simon, David Chisnall, and Ross J. Anderson. What you get is what you C: controlling side effects in mainstream C compilers. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 1–15. IEEE.

[63] David M Sommer, Sebastian Meiser, and Esfandiar Mohammadi. Privacy loss classes: The central limit theorem in differential privacy. *Proceedings on Privacy Enhancing Technologies*, 2:245–269, 2019.

[64] Samuel Weiser, David Schrammel, Lukas Bodner, and Raphael Spreitzer. Big numbers - big troubles: Systematically analyzing nonce leakage in (EC)DSA implementations. In *29th USENIX Security Symposium*, pages 1767–1784. USENIX Association.

[65] Jan Wichelmann, Ahmad Moghimi, Thomas Eisenbarth, and Berk Sunar. Microwalk: A framework for finding side channels in binaries. In *Proceedings of the 34th Annual Computer Security Applications Conference*, page 161–173, New York, NY, USA, 2018. Association for Computing Machinery.

[66] Jan Wichelmann, Florian Sieck, Anna Pätschke, and Thomas Eisenbarth. Microwalk-ci: Practical side-channel analysis for javascript applications. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022*. ACM, 2022.

[67] Luca Wilke, Florian Sieck, and Thomas Eisenbarth. Tdx-down: Single-stepping and instruction counting attacks against intel TDX. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 79–93. ACM, 2024.

[68] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656, 2015.

[69] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. *IEEE Transactions on Computers*, 69(1):126–137, 2020.

[70] Yuqing Zhu and Yu-Xiang Wang. Poisson subsampled rényi differential privacy. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 7634–7642. PMLR, 09–15 Jun 2019.

Algorithm 4: TrainSBDT : Train a DP GBDT ensemble

```
Input: $D$ : training data, $\gamma$ : subsampling ratio
       $\varepsilon_{\text{init}}$ : privacy budget for initial score
       $(\varepsilon_{\text{trees}}, \delta_{\text{trees}})$ : DP parameters for training of trees
       $(r_1, r_2)$ : noise weights for leaf value
       $(g^*, h^*, m^*)$ : gradient/Hessian/label clipping bound
       $\lambda, \beta$ : regularization parameters
       $(T_{\text{regular}}, T_{\text{extra}})$ : number of regular and extra rounds
       $\alpha_{\max}$ : largest $\alpha$ to test in Rényi DP, $d$ : depth of trees
1  $T_{\max} = T_{\text{regular}} + T_{\text{extra}}$
2  $\hat{\alpha}, \sigma^2_{\text{leaf}}, \rho(\hat{\alpha}) = \texttt{Initialize}(\alpha_{\max}, (\varepsilon_{trees}, \delta_{trees}), \varepsilon_{init}, \gamma)$
3  $\text{init}_0 = \texttt{DPInitScore}(D, m^*, \varepsilon_{init})$
4  $E = (\text{init}_0)$
5  for $t = 1$ to $T_{max}$ do
6      for $i = 1$ to $|D|$ do
7          $\rho_t^{(i)}(\alpha) := a_\gamma(\alpha, \frac{\alpha}{\sigma^2_{\text{leaf}}} \cdot \left(\frac{r_1 \cdot |h_i|^2}{(h^*)^2} + \frac{r_2 \cdot |g_i|^2}{(g^*)^2}\right))$
8      $D_t = (x_i : \mathcal{F}_{\hat{\alpha}, \rho(\hat{\alpha})}(\rho_1^{(i)}, \ldots, \rho_t^{(i)}) = \text{CONT})$
9      $\text{tree}_t = \texttt{TrainSingleTree}(D_t, d, \sigma^2_{leaf}, g^*, h^*, (r_1, r_2), \lambda, \beta, E)$
10     $E = (\text{init}_0, \text{tree}_1, ..., \text{tree}_t)$
11 return E
```

A Ablation study on early tree stopping

In Figure 4 we provide an ablation study on the abalone dataset for a varying number of maximal number of trees. We observe that for 50 trees, early stopped S-BDT performs on par for low ε compared to the HP-tuned an non-HP-tuned variant but fail to reaches a good RMSE for higher ε. Choosing 600 trees allows early stopped S-BDT to capture a good utility for a high ε while at the same time performing much better than non-HP-tuned S-BDT and within the vicinity of leaky HP-tuned S-BDT

B Pseudocode for Constant-Time S-BDT

C Postponed proofs

Theorem 9. *Algorithm 3 satisfies* $(1 \mapsto \rho_{input}(\alpha))$*-Timing Privacy.*

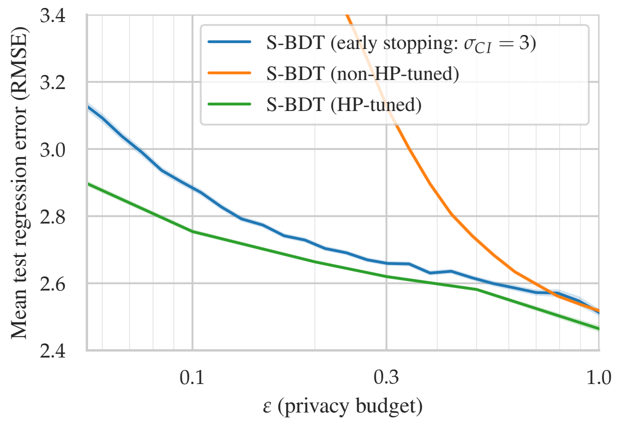
Proof. Let $X \sim X'$ be two neighboring data sets and $y \in \mathcal{R}$ an arbitrary output. Let $T_{\max}$ be the number of trees, d the depth of trees, $\alpha_{\max} \cdot \sigma_{\max}$ the number of iterations performed

Algorithm 5: DPInitialScore : Compute a DP initial score

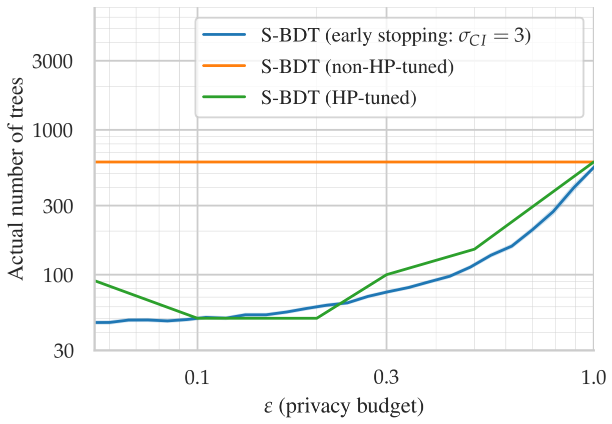
```
Input: $D$ : training dataset, $m^*$ : clipping bound on labels
       $\varepsilon_{\text{init}}$ : DP privacy budget for initial score
       $\varepsilon_{\text{ds}} = 0.025$: DP privacy budget for dataset size
1  $|D|_{\text{priv}} = |D| + \texttt{Gauss}(0, 2 \cdot \ln(1.25/\delta)/\varepsilon^2_{ds})$
2  if regression then
3      $M = \frac{1}{|D|_{\text{priv}}} \sum_{y_i \in D} \texttt{clamp}(y_i, -m^*, m^*)$
4      $M_{\text{priv}} = M + \texttt{Gauss}(0, 2 \cdot \ln(1.25/\delta)(m^*/(|D|_{priv} \cdot \varepsilon_{init}))^2)$
5      return $M_{\text{priv}}$
6  else if classification then
7      $M = \frac{1}{|D|_{\text{priv}}} \sum_{y_i \in D} \texttt{clamp}(y_i, 0, m^*)$
8      $M_{\text{priv}} = M + \texttt{Gauss}(0, 2 \cdot \ln(1.25/\delta)(m^*/(|D|_{priv} \cdot \varepsilon_{init}))^2)$
9      return $\ln(M_{\text{priv}}/1 - M_{\text{priv}})$
```

Algorithm 6: TrainSingleTree: Train a DP decision tree

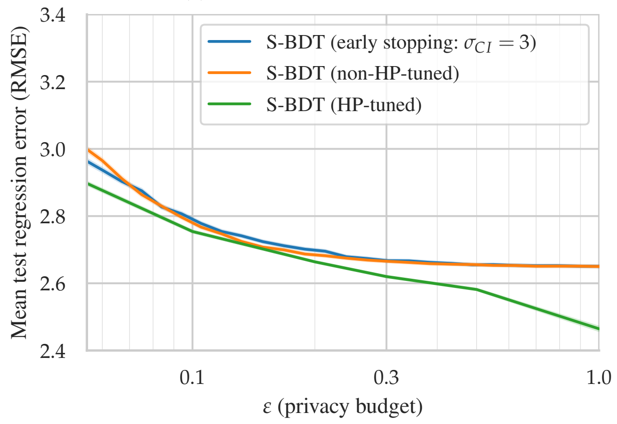
```
Input: $D$ : training data, $d$ : depth of trees
       $\sigma^2_{\text{leaf}}$ : unweighted variance of Gaussian for leaves
       $(g^*, h^*)$ : clipping bound on gradients and Hessians
       $(r_1, r_2)$ : noise weights for leaf value
       $\lambda, \beta$ : regularization parameters
       $E$ : ensemble of trees up to round $t - 1$
1  $\mathcal{P} = \texttt{PredictEnsemble}(D_t, E, \eta)$
2  $\texttt{ComputeGradientsHessians}(\mathcal{P}, D_t)$
3  $\texttt{PoissonSubsample}(D, \gamma)$;
4  $\text{tree}_t = \texttt{RandomTree}$ (d)
5  for each leaf $l$ in tree$_t$ do
6      $v = \texttt{DPLeaf}(l, D, g^*, h^*, \sigma^2_{leaf}, (r_1, r_2), \lambda, \beta)$;
7      $\texttt{SetLeaf}$ $(\text{tree}_t, l, v)$;
8  return tree$_t$;
```



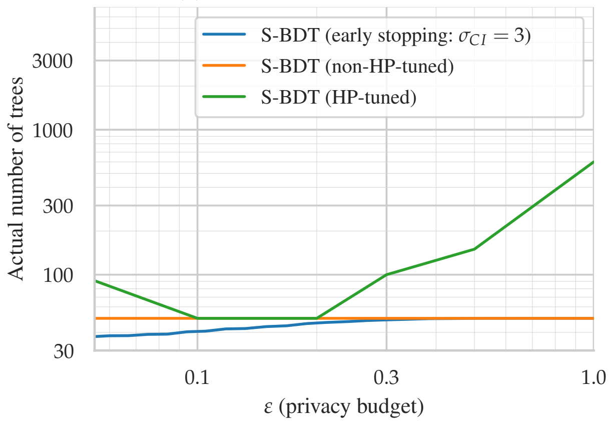
(a) maximal $T = 600$ trees



(b) maximal $T = 600$ trees



(c) maximal $T = 50$ trees



(d) maximal $T = 50$ trees

Figure 4: S-BDT on abalone [48] with privacy-leaky hyperparameter-tuning (green), with statically chosen hyperparameters (orange), or with early stopping (blue). We plot the privacy-utility tradeoff (left) and the selected number of trees (right). Evaluated on 200-repeated 5-fold cross-validation.

Algorithm 7: DPLeaf : Compute a DP leaf node

```
Input: $D$ : training data, $l$: leaf node identifier
       $(g \in D, h \in D)$ : gradient/Hessian of data points in $D$
       $(g^*, h^*)$ : clipping bound on gradients and Hessians
       $(r_1, r_2)$ : noise weights for leaf value
       $\lambda, \beta$ : regularization parameters
       $\sigma^2_{\text{leaf}}$ : unweighted variance of Gaussian for leaves
1 $w = \sum_{h \in D} \texttt{Clamp}(h, 0.0, h^*) \cdot \mathbb{1}\{\text{data point is in leaf } l\}$
2 $\tilde{w} = \lambda + w + \texttt{Gauss}(0.0, (h^*)^2 \sigma^2_{leaf} / (2 \cdot r_1))$
3 $u = \sum_{g \in D} \texttt{Clamp}(g, -g^*, g^*) \cdot \mathbb{1}\{\text{data point is in leaf } l\}$
4 $\tilde{u} = u + \texttt{Gauss}(0.0, (g^*)^2 \sigma^2_{leaf} / (2 \cdot r_2))$
5 return $\texttt{Clamp}(v = \tilde{u} / \tilde{w}, -\beta, \beta)$
```

in Algorithm 8. Let $\tilde{l} := |X| + \mathcal{N}(0, 1/\varepsilon_{\text{input}})$ be the DP approximated input length of input X.

By direct analysis we get the following components of Algorithm 3 and their runtime: Gaussian sampling and noising input length (2 time steps), Algorithm 8 ($\alpha_{\max} \cdot \sigma_{\max} \cdot 16$), Algorithm 5 ($\tilde{l} \cdot 2 + 6$), in Algorithm 4 updating individual privacy budgets ($\tilde{l} \cdot (11 + (\alpha_{\max} - 1) \cdot 11 + 12)$) and filtering ($\tilde{l}$), Algorithm 9 ($\tilde{l}$), Algorithm 10 ($(2^d - 1 + 2^d \cdot 2 + 2)$, Algorithm 11 ($\tilde{l} \cdot 2$), Algorithm 12 ($\tilde{l} \cdot 2$), generating uniformly random splits for the random tree ($2^{d+1} - 1$), Algorithm 7 ($2^d \cdot (\tilde{l} \cdot 3 + 3 + \tilde{l} \cdot 3 + 2 + 1$) and writing output back to memory ($2^{d+1} - 1) \cdot T_{\max}$). We observe a constant amount of operations independent of input length

$$K = 2 \cdot \left(4 + 8\alpha_{\texttt{max}}\sigma_{\texttt{max}} - T_{\max} + 5 \cdot 2^d T_{\max}\right)$$

and a constant amount of iterations for each data point in the

Algorithm 8: Initialize Rényi DP accountant

```
Input: $\alpha_{\texttt{max}}$ : largest $\alpha$ to test in RDP, $\gamma$ : subsampling ratio
       $(\varepsilon_{\text{trees}}, \delta_{\text{trees}})$ : DP parameters for training of trees
       $\varepsilon_{\text{init}}$ : privacy budget for DP initial score
1 $\mathcal{T} = ()$
2 for $\alpha = 2$ to $\alpha_{max}$ do
3     for $\sigma^2_{leaf}$ in $(0.0, 1000.0]$ do
4         $\rho_{\text{subsampled-tree}} = a_\gamma(\alpha, \frac{\alpha}{\sigma^2_{\text{leaf}}})$
          $\rho(\alpha) = T_{\text{regular}} \cdot \rho_{\text{subsampled-tree}}$
          $\varepsilon'_{\text{trees}} = \rho(\alpha) + \log\frac{\alpha-1}{\alpha} - \frac{\log\delta_{\text{trees}} + \log\alpha}{\alpha-1}$
5         Append($\mathcal{T}, (\alpha, \sigma^2_{leaf}, \rho(\alpha), \varepsilon'_{trees})$)
6 Pick $(\hat{\alpha}, \sigma^2_{\text{leaf}}, \rho(\hat{\alpha}), \varepsilon'_{\text{trees}})$ from $\mathcal{T}$ so that $\sigma^2_{\text{leaf}}$ is
    smallest and $\varepsilon'_{\text{trees}}$ is close to $\varepsilon_{\text{trees}}$
7 Report $\varepsilon_{\text{init}} + \varepsilon'_{\text{trees}}$ to user
8 return $\hat{\alpha}, \sigma^2_{\text{leaf}}, \rho(\hat{\alpha})$
```

Algorithm 9: PredictEnsemble : Compute prediction of ensemble

```
Input: $D$ : data
       $E$ : ensemble of trees
       $\eta$ : learning rate
1 $\mathcal{P} = ()$
2 if ensemble has initial score then
3     for each data point $x$ in $D$ do
4         Let $i$ be the index of $x$
5         $\mathcal{P}[i] = E[0]$
6 for each data point $x$ in $D$ do
7     Let $i$ be the index of $x$
8     for each tree $t$ in $E$ do
9         Let $r$ be the root of $t$
10        $\mathcal{P}[i] = \mathcal{P}[i] + fdsayx\eta \cdot \texttt{PredictTree}(r, x)$
```

Algorithm 10: PredictTree : Compute prediction of tree

```
Input: $n$ : node of tree
       $x$ : data point
1 if $n$ is leaf then
2     return prediction of $r$
3 Let $l, r$ be the left and right children of $n$
4 left-prediction = $\texttt{PredictTree}(l, x)$
5 right-prediction = $\texttt{PredictTree}(r, x)$
6 Let $j, a$ be split attribute and split value of $n$
7 if $x[j] \geq a$ then
8     return right-prediction
9 else
10    return left-prediction
```

Algorithm 11: ComputeGradientsHessians : Compute gradients and Hessians

```
Input: $\mathcal{P}$ : ensemble prediction on $D$
       $D$ : data
1 if regression then
2     for each data point $x_i$ in $D$ do
3         $g_i = \mathcal{P}[i] - y_i$
4         $h_i = 1$
5         Save gradient and Hessian to $D$
6 else if classification then
7     for each data point $x_i$ in $D$ do
8         $g_i = 1/(1 + \exp(-\mathcal{P}[i])) - y_i$
9         $h_i = (y_i + g_i) \cdot (1 - y_i - g_i)$
10        Save gradient and Hessian to $D$
```

Algorithm 12: PoissonSubsample : Generate a Poisson subsample

```
Input: $D$ : data
       $\gamma$ : subsampling ratio
1 for each data point $x_i$ in $D$ do
2     $b = \texttt{BernoulliBit}$
3     if $b == 1$ then
4         Set $x_i$ active in current round
5     else if $b == 0$ then
6         Set $x_i$ inactive in current round
```

input

$$k = 2 + \frac{(35 + 9 \cdot 2^d + 22\alpha_{\mathtt{max}})T_{\mathtt{max}}}{2} + \frac{(1 + 3 \cdot 2^d)T_{\mathtt{max}}^2}{2}$$

We compute the Timing Privacy guarantee

$$\begin{aligned}
&D_\alpha(T_M(X,\text{env})|_{\text{out}(M(X,\text{env}))=y}|| \\
&T_M(X',\text{env}')|_{\text{out}(M(X,\text{env}))=y}) \\
&= D_\alpha(\mathcal{N}(|X| + K, k^2 \cdot 1/\varepsilon_{\text{input}})|| \\
&\mathcal{N}(|X'| + K, k^2 \cdot 1/\varepsilon_{\text{input}})) \\
&= D_\alpha(\mathcal{N}(0, k^2 \cdot 1/\varepsilon_{\text{input}})||\mathcal{N}(1, k^2 \cdot 1/\varepsilon_{\text{input}}))
\end{aligned}$$

Using [46, Proposition 7]

$$\leq \frac{\alpha}{2 \cdot k^2 \cdot 1/\varepsilon_{\text{input}}} =: \rho_{\text{input}}(\alpha)$$

$\square$