SPARSE FOREST: EFFICIENT ONLINE LEARNING AND UNLEARNING FRAMEWORK FOR TREE ENSEMBLES

Anonymous authors
Paper under double-blind review

ABSTRACT

Evolving data and limited computational resources pose two coupled challenges for deploying machine learning system in reality. Online learning and unlearning enable models to incorporate new observations and remove obsolete records, while the computational efficiency of tree ensembles makes them attractive for resource-constrained deployment. However, inexpensive training and inference do not imply inexpensive maintenance: a single sample insertion or deletion can trigger costly whole tree reconstruction. We introduce *Sparse Forest* that addresses this bottleneck through three complementary forms of sparsity. *Structural sparsity* limits the complexity of the maintained trees, reducing the fundamental cost of node statistics updates. *Reconstruction sparsity* prunes the reconstruction through in-place subtree repair, reusing existing descendants and propagating changes in sample membership. *Split-change sparsity* reduces the frequency of structural changes by retaining satisfactory splits under a controlled suboptimality tolerance. Together, these principles control how much structure must be maintained, how much work each repair requires, and how often repairs necessiates. By formalizing (ε, φ)-suboptimality for tree ensembles, Sparse Forest makes exactness an explicit, configurable property rather than a fixed design choice. At $\varepsilon = 0$, Sparse Forest maintains a model that is *bit-identical* to retraining from scratch on the updated dataset. To our knowledge, this is the first such guarantee for online learning and unlearning in tree ensembles, going beyong equality in distribution. Relaxing this condition ($\varepsilon > 0$) trades retraining equivalence for substantially fewer structural changes at negligible accuracy cost. Experiments demonstrate orders-of-magnitude speedups over the state-of-the-art online learning and unlearning baseline, establishing Sparse Forest as a practical alternative for real-world deployment. Code is available in our anonymous repository.

1 INTRODUCTION

Deployed *machine learning* (ML) systems rarely operate on a fixed dataset in real world scenarios. New observations arrive, existing records are updated or corrected, outdated samples are retired, while the system must continue to serve predictions within a limited computational budget. Retraining the model after every such change keeps the model faithful to the data, but it is prohibitively expensive when changes are frequent, particularly when maintenance competes with the prediction workload for the same resource. These demands motivate *online learning and unlearning* or online (un)learning for short: maintaining a model under an interleaved stream of sample insertions, sample deletions, and prediction queries. Rather than treating training as a one-time stage before depolyment, online (un)learning performs small, targeted in-place adjustments that *ideally* produces an model that is indistinguishable from one retrained on the updated data. How closely this ideal must be met, and at what cost, are the central problem and trade-off of online (un)learning.

Tree ensembles, particularly *Random Forests* (RFs) (Breiman, 2001), *Extremely Randomized Trees* (ERTs) (Geurts et al., 2006), are natrual candidates for this setting, for two reasons. First, they peform strongly on tabular data, are inexpensive to train and depoly, and are consequently widespread in application such as financial risk assessment (Kruppa et al., 2013), medical diagnosis (Casanova et al., 2014), recommendation (Zhang & Min, 2016), and on-device network intrusion detection (Hamdouchi & Idri, 2025), where models are often served from CPU-only servers or edge devices rather than from accelerators. Second, in these tree ensembles, the influence of a single sample can

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

be localized and exactly attributable (down to a concrete tree path), since each of its decisions is a discrete function of an indentifiable subset of the training samples. Thus, the tree structure not only bring interpertability but also makes exact in-place sample insertion and deletion possible for tree ensembles, which is unavailable to gradient-based methods, where the impact of a single sample could affect possibly all parameters and is aggregated and entangled through iterative optimization.

However, the same structure that makes exact adjustment possible, also makes it costly, as local data changes do not necessarily have local structural consequences. Inserting or deleting a single sample can change the split installed at a node, and replacing that split changes the samples routed to its descendents and invalidate the whole subtree in return. Existing methods respond by discarding the affected subtree and regrowing it, which become the bottleneck of the efficiency. Efficient online (un)learning for trees thus requires solving two problems at once: lowering the frequency of installed splits change, and reducing the structural consequence of such change.

We propose *Sparse Forest* (SF), a framework for online (un)learning in tree ensembles that alleviate these problems through three complementary forms of sparsity. *Structural sparsity* limits the number of sample contained in each tree, and make the whole tree structure / node slim. *Reconstruction sparsity* repairs the invalidated subtree in-place by transferring samples between existing children and recusively updating states affected by transfers, taking the most advantage / reusing of the statistics maintained in the invalidated subtree; *Split-change sparsity* reduces the frequency of split replacements by relaxing the the optimal condition to the proposed (ε, φ)-suboptimality condition. Our main contributions are summarized as follows:

Efficient online (un)learning framework. We propose SF for handling online (un)learning request stream, integrating structural, reconstruction, split-change sparsity. We provide algorithms with implementation and analyze the time and memory complexity of its maintenance mechanisms.

Configurable exactness. We formalize (ε, φ)-suboptimality, which allows a score gap of $\varepsilon n^{-\varphi}$ between the installed split and the optimal split at a node containing n samples. At $\varepsilon = 0$, SF achieves *total exactness*: the maintained model is bit-identical to the retrained model. For $\varepsilon > 0$, SF sacrifices this equivalence for efficiency, making exactness a configurable property.

Empirical evaluation. We compare SF with existing relevant models, and results show that SF with $(\varepsilon, \varphi) = (\frac{1}{10}, \frac{1}{2})$ is $17.5\times$ ($81\times$ in the largest dataset) faster than previous sota model request-processing time while retaining competitive predictive performance and using only 38% of memory. The total exact SF with $\varepsilon = 0$ is still $4\times$ faster than the previous fastest model.

2 Background

This section fixes notation, formalizes the problem, and position Sparse Forest relative to prior work.

Let $\mathcal{Z} \subseteq \mathcal{X} \times \mathcal{Y}$ be the sample space, where $\mathcal{X}$ is the attribute (input) space (e.g., $\mathcal{X} \subseteq \mathbb{R}^d$ or $\mathcal{X} \subseteq [-1, 1]^d$) and $\mathcal{Y}$ is the label space (e.g., $\mathcal{Y} = \{-1, 1\}$ for binary classification or $\mathcal{Y} = \mathbb{R}$ for regression). A training set (dataset) $D \subseteq \mathcal{Z}$ is a finite set of samples $\mathbf{z}_i = \langle \mathbf{x}_i, y_i \rangle \in \mathcal{Z}$. A possibly randomized training algorithm $\mathcal{A}$ maps training set D to a model $\mathsf{M} = \mathcal{A}(D)$, and the model M maps a query input $\mathbf{x}_q \in \mathcal{X}$ to a prediction $\hat{y}_q = \mathsf{M}(\mathbf{x}_q) \in \mathcal{Y}$. We call a model accurate if $\hat{y}_q$ is close to the true label $\mathbf{y}_q$, under a task-specific criterion, with high probability.

2.1 Online Learning and Unlearning

Online (un)learning. *Online learning* can be formalize as an algorithm $\mathsf{M}' = \mathcal{A}_+(\mathsf{M}, \mathbf{z})$ that maps model M and a new sample $\mathbf{z}$ to an updated model M', and similarily, *online unlearning* is an algorithm $\mathsf{M}' = \mathcal{A}_-(\mathsf{M}, \mathbf{z})$ for some $\mathbf{z}$ already exist in the dataset. In the first case, M' should incorporate the influence of $\mathbf{z}$ as if $\mathbf{z}$ had been part of the training set D, i.e., M' should matches $\mathsf{M}_{\mathrm{re}} = \mathcal{A}(D \cup \{\mathbf{z}\})$; while for unlearning, M' should be free of the influence of $\mathbf{z}$, and matches $\mathsf{M}_{\mathrm{re}} = \mathcal{A}(D \setminus \{\mathbf{z}\})$. What *matches* means is deliberately left open at this point, and we will make it precise after fixing the setting in which these algorithms are applied.

For tree-based online learning, VFDT (Domingos & Hulten, 2000) incrementally grows DTs from data streams, Manapragada et al. (2018) proposes EFDT that changes already installed splits for adaptivity, while Lourenço et al. (2026) adapts tree growth to memory constraints. For unlearning,

the model-agnostic SISA framework (Bourtoule et al., 2021) limits retraining through sharding and checkpointing, whereas DaRE (Brophy & Lowd, 2021) and HedgeCut (Schelter et al., 2021) tailor randomized tree ensembles to efficient sample deletion. Lately, DynFrs (Wang et al., 2025) support both online learning and unlearning, and use subsampling to reduce workload, and Lin et al. (2025) introduces online (un)learning framework for Gradient Boosting Decision Trees (GBDTs).

Request stream. During deployment, the ML framework will receive an interleaved sequence of requests for sample insertion, sample deletion, and prediction query. Starting from the initial training set $D_0 = D$, at each time step $t \geq 1$, the model receives a request $\mathbf{r}_t = \langle \mathsf{o}_t, \mathbf{v}_t \rangle$, where $\mathsf{o}_t \in \{+, -, \varnothing\}$ is the request type and $\mathbf{v}_t$ is its argument, either a sample $\mathbf{z}_t$ or an input $\mathbf{x}_t$. Concretely, $\mathbf{r}_t$ is one of the following: $\mathsf{Insert}(\mathbf{z}_t) = \langle +, \mathbf{z}_t \rangle$, $\mathsf{Delete}(\mathbf{z}_t) = \langle -, \mathbf{z}_t \rangle$, $\mathsf{Query}(\mathbf{x}_t) = \langle \varnothing, \mathbf{x}_t \rangle$. Writing $\mathcal{A}_\pm(\mathsf{M}, \mathbf{r}_t) \triangleq \mathcal{A}_{\mathsf{o}_t}(\mathsf{M}, \mathbf{v}_t)$ as the *online (un)learning algorithm*, and setting $\mathcal{A}_\varnothing(\mathsf{M}, \mathbf{x}) = \mathsf{M}$ for convenience (because a query leaves the training set and model unchanged). At time step t, the request $\mathbf{r}_t$ updates the dataset D_{t-1} and model M_{t-1} as:

$$D_t = \begin{cases} D_{t-1} \cup \{\mathbf{z}_t\} & \mathsf{o}_t = + \\ D_{t-1} \setminus \{\mathbf{z}_t\} & \mathsf{o}_t = - \\ D_{t-1} & \mathsf{o}_t = \varnothing \end{cases}; \qquad \mathsf{M}_t = \mathcal{A}_\pm(\mathsf{M}_{t-1}, \mathbf{r}_t) = \begin{cases} \mathcal{A}_+(\mathsf{M}_{t-1}, \mathbf{z}_t) & \mathsf{o}_t = + \\ \mathcal{A}_-(\mathsf{M}_{t-1}, \mathbf{z}_t) & \mathsf{o}_t = - \\ \mathcal{A}_\varnothing(\mathsf{M}_{t-1}, \mathbf{x}_t) & \mathsf{o}_t = \varnothing \end{cases}$$

Specially, a query (request with $\mathsf{o}_t = \varnothing$) additionally returns $\hat{y}_t = \mathsf{M}_t(\mathbf{x}_t)$ as the prediction.

Exactness. The natural reference for the maintained model after t requests is the retraing from scratch model on the current data, which is $\mathsf{M}_{\mathrm{re}} = \mathcal{A}(D_t)$. Conventionally (Brophy & Lowd, 2021; Wang et al., 2025), the *matching* or *closeness* between the maintained model M_t produced by $\mathcal{A}_\pm$ and the retrained model M_{re} produced by $\mathcal{A}$ are measured by equality in distribution of models.

Definition 1: Distributional Exactness
The online (un)learning algorithm $\mathcal{A}_\pm$ is *distributionally exact* with respect to the training algorithm $\mathcal{A}$ if, for every request stream and every $t \geq 1$, the maintained model $\mathsf{M}_t = \mathcal{A}_\pm(\mathsf{M}_{t-1}, \mathbf{r}_t)$ (with $\mathsf{M}_0 = \mathcal{A}(D_0)$) and the freshly retrained model $\mathsf{M}_{\mathrm{re}} = \mathcal{A}(D_t)$ have the same distribution over the model space $\mathcal{M}$:

$$\forall\, t \geq 1,\ \forall\, \mathsf{M} \in \mathcal{M}: \quad \Pr[\mathsf{M}_t = \mathsf{M}] = \Pr[\mathsf{M}_{\mathrm{re}} = \mathsf{M}].$$

In this study, we propose and pursue a stricter condition:

Definition 2: Total Exactness
For determinsitic algorithms or random algorithm with the same specification, we say such $\mathcal{A}_\pm$ is totally exact to $\mathcal{A}$ if, for every request stream and every $t \geq 1$, $\mathsf{M}_t = \mathcal{A}_\pm(\mathsf{M}_{t-1}, \mathbf{r}_t)$ (with $\mathsf{M}_0 = \mathcal{A}(D_0)$) and $\mathsf{M}_{\mathrm{re}} = \mathcal{A}(D_t)$ are bit-identical to each other:

$$\forall\, t \geq 1: \qquad \mathsf{M}_t = \mathsf{M}_{\mathrm{re}}.$$

Clearly, total exactness implies distribution exactness because equality holds for every realization of the shared randomness. The term *bit-identical* means that for all nodes in both models, the node sample set D_u, installed splits $\mathbf{s}_u$, leaf statistics, and all components that logically constitute the tree ensembles are the same for both models, correspondingly.

2.2 Suboptimal Tree Ensembles

A *Decision Tree* (DT) recursively partitions the attribute space $\mathcal{X}$ by splits predicates at internal nodes, and returns predictions from the sample statistics stored at the leaf that the query reaches. For an internal node u, let $D_u \subseteq D$ be the set of samples routed to node u, and $n_u = |D_u|$ be the number of samples. A numerical split $\mathbf{s} = \langle \alpha, \theta \rangle$ with attribute index α and threshold θ routes a sample $\langle \mathbf{x}, y \rangle$ to the left child if $\mathbf{x}_\alpha < \theta$ and to the right child otherwise, inducing the partition $D_u^{\mathrm{L}}(\mathbf{s}) = \{\mathbf{z}_i = \langle \mathbf{x}_i, y_i \rangle \in D_u \mid \mathbf{x}_{i,\alpha} < \theta\}$, and $D_u^{\mathrm{R}}(\mathbf{s}) = D_u \setminus D_u^{\mathrm{L}}(\mathbf{s})$. The installed split $\hat{\mathbf{s}}_u$ at node u is chosen from a finite split candidate set C_u by optimizing the *split criterion* $\mathcal{I}(D_u, \mathbf{s})$:

$$\mathcal{I}(D_u, \mathbf{s}) = \sum_{b \in \{\mathrm{L},\mathrm{R}\}} \frac{|D_u^b(\mathbf{s})|}{|D_u|} \cdot \mathcal{H}\big(D_u^b(\mathbf{s})\big), \quad \text{where } \mathcal{H}(D) = \begin{cases} 1 - \sum_{y \in \mathcal{Y}} \hat{p}_D(y)^2 & \text{Gini index} \\ -\sum_{y \in \mathcal{Y}} \hat{p}_D(y) \log \hat{p}_D(y) & \text{Entropy} \end{cases}$$

with $\mathcal{H}$ being an impurity measure and $\hat{p}_D(y)$ being the fraction of samples in D with label y. Greedy induction installs the empirically optimal split $\mathbf{s}_u^\star \in \arg\min_{\mathbf{s}\in C_u} \mathcal{I}(D_u, \mathbf{s})$, and routes D_u to two children according to $\mathbf{s}_u^\star$, and recurses until a stopping rule declares a leaf.

Prior work relaxes greedy split selection in different ways. VFDT (Domingos & Hulten, 2000) commits to a split once its advantage over the runner-up exceeds a confidence radius proportional to $1/\sqrt{n_u}$, and fully dynamic DTs (Bressan et al., 2023; Bressan & Sozio, 2024) accept any split within a constant additive tolerance of the optimal. We unify constant and sample-size-dependent additive split-quality requirements within a single family:

Definition 3: (ε, φ)-Suboptimality

Fix $\varepsilon \geq 0$ and $\varphi \geq 0$. A tree ensemble is (ε, φ)-*suboptimal* if, for any node u in the model, with $n = n_u = |D_u|$ being the number of samples in the node, the installed split $\hat{\mathbf{s}}_u$ satisfies that

$$\mathcal{I}(D_u, \hat{\mathbf{s}}_u) - \min_{\mathbf{s}\in C_u} \mathcal{I}(D_u, \mathbf{s}) \leq \frac{\varepsilon}{n^\varphi} = \varepsilon n^{-\varphi}. \tag{1}$$

Here, ε controls the magnitude of the tolerance and φ controls its sample-size dependency. Smaller ε tightens the condition, and for fixed $\varepsilon > 0$, larger φ further tightens it at nodes with $n > 1$. Setting $\varepsilon = 0$ recovers empirical split optimality, whereas positive tolerance permits retaining a non-optimal split. The $(\varepsilon, 0)$ case recovers the constant additive split-quality requirement studied in fully dynamic DTs (Bressan et al., 2023; Bressan & Sozio, 2024), under the same criterion and candidate set. The $\varphi = \frac{1}{2}$ case *mirros* the inverse-square-root confidence threshold used by EFDT (Manapragada et al., 2018), which, unlike VFDT, reevaluates installed splits. However, the Hoeffding condition used by EFDT still does not guarantee $(\varepsilon, \frac{1}{2})$-suboptimality.

3 Methods

In this section, we introduce the three forms of sparsity used by Sparse Forest: *Structural sparsity* (Section 3.1 and Fig. 1) fixes a per-node representation that is cheap to build and update, and limits the number of trees a sample affects. *Reconstruction sparsity* (Section 3.2 and Fig. 2) makes in-place repair instead of reconstructing the subtree when the installed split changes. *Split-change sparsity* (Section 3.3 and Fig. 3) sparsifies replacements by retaining splits that remains (ε, φ)-suboptimal, and certifies this property with $\mathcal{O}(1)$ work per request.

3.1 Structural Sparsification

Light-weight split maintenance. Sparse Forest reduces the complexity of tree structure maintenance using a light weight split organization algorithm. At node u, the installed split is chosen from a candidate set C_u with ps splits in it, where $p = \lceil \sqrt{d} \rceil$ is the number of attributes, which is chosen randomly out of the d attributes, and s is the number of thresholds considered for each such attribute. Sparse Forest places the s thresholds uniformly instead of randomly for better time complexity. Let $[l_{u,\alpha}, r_{u,\alpha}]$ be the range prescribed by the construction rule, we define $\Delta_{u,\alpha} \triangleq (r_{u,\alpha} - l_{u,\alpha})/(s+1)$ as the even gap, $\theta_{u,\alpha,j} = l_{u,\alpha} + j\Delta_{u,\alpha}$ as the j-th threshold of attribute α.

The uniform threshold grid allow all thresholds of an attribute to share a single histogram. For a fixed node u and fixed attribute α (we further suppress the node index and attribute index in the subscript for simplicity), the thresholds $\theta_{1\ldots s}$ divides $[l, r]$ into $s+1$ bins, indexed by $b = 0, 1, \ldots, s$. A sample $\langle \mathbf{x}, y \rangle$ belongs to the $b(\mathbf{x})$-th bin, where $b(\mathbf{x}) = \lfloor (\mathbf{x}_\alpha - l)/\Delta \rfloor$. Then, we maintain bin counts $c_{0\ldots s}$ and bin counts $c_{y,0\ldots s}$ for class $y \in \mathcal{Y}$. An intuitive $\mathcal{O}(n)$ (for $n = |D|$) scan reveal the bin counts over sample set D by doing two operations per sample (Fig. 1 upper-right):

$$\forall \langle \mathbf{x}_i, y_i \rangle \in D: \qquad \beta \leftarrow b(\mathbf{x}_i) = \lfloor (\mathbf{x}_{i,\alpha} - l)/\Delta \rfloor, \quad c_\beta \leftarrow c_\beta + 1, \quad c_{y_i,\beta} \leftarrow c_{y_i,\beta} + 1.$$

Furthermore, the bin counts can be updated in $\mathcal{O}(1)$ when a sample is added to or deleted from D. We then show that the bin counts for attribute α are essential that it can derive split scores $\mathcal{I}(D, \mathbf{s})$ for all splits $\mathbf{s}_{1\ldots s} = \langle \alpha, \theta_{1\ldots s} \rangle$ in $\mathcal{O}(s)$. For split $\mathbf{s}_j = \langle \alpha, \theta_j \rangle$, let $n^{\mathrm{L}} = |D^{\mathrm{L}}(\mathbf{s})|$ be the number of samples that would be split to left subtree, and n^{R} be that number of right subtree. We then define $n_y^{\mathrm{L}} = |\{\langle \mathbf{x}_i, y_i \rangle \in D^{\mathrm{L}}(\mathbf{s}) \mid y_i = y\}|$ be the number of samples splits to the left and have label y, and define n_y^{R} in a similar way. These qunatities suffice to evaluate the Gini index or Entropy score

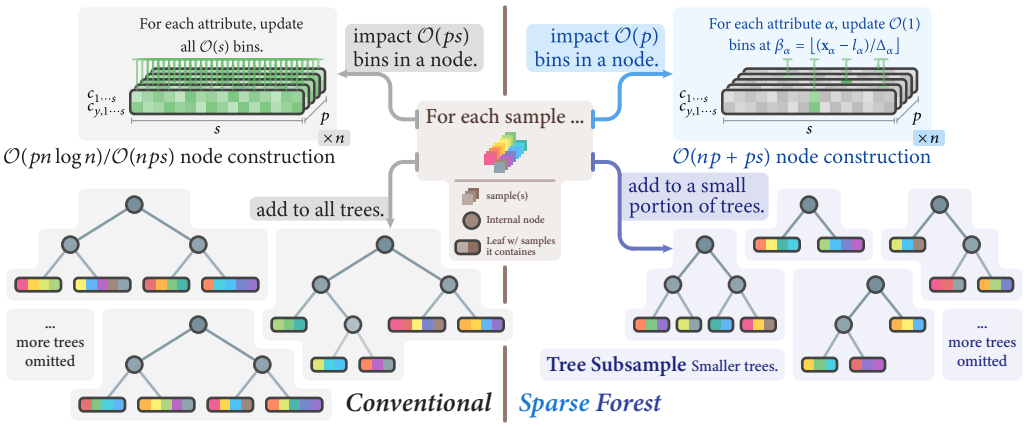


Figure 1: ***Structural Sparsity***: Sparse Forest maintains node statistics with lower time complexity, and adopt tree subsampling to reduce workload and tree size. **Left**: The conventional way to maintain node statistics and find the optimal split, which is $\mathcal{O}(pn \log n)$ or $\mathcal{O}(nps)$, and conventional way adds each sample to all trees in the forest. **Right**: Sparse Forest use an efficient maintenance that construct a node in $\mathcal{O}(np + ps)$, and it adds each sample to only 20% of the trees in the forest.

$\mathcal{I}(D_u, \mathbf{s}_j)$ (see Appendix B.1, Eq. 5). Here, the necessary quantity n^{L}, n^{R}, n_y^{L}, and n_y^{R}, which is known as the *node statistics*, can be obtained in $\mathcal{O}(1)$:

$$n^{\mathrm{L}} = c'_{j-1} = \sum_{\beta=0}^{j-1} c_\beta, \qquad n^{\mathrm{R}} = n - n^{\mathrm{L}}, \qquad n_y^{\mathrm{L}} = c'_{y,j-1} = \sum_{\beta=0}^{j-1} c_{y,\beta}, \qquad n_y^{\mathrm{R}} = n_y - n_y^{\mathrm{L}}.$$

if we obtain $c'_{0\cdots s}$ and $c'_{y,0\cdots s}$ as the prefix sum of $c_{0\cdots s}$ and $c_{y,0\cdots s}$ before evaluating the split score. Proof of equivalence are available in Appendix B.1. Thus, for fixed attribute $alpha$, obtaining the bin counts needs a $\mathcal{O}(n)$ scan, deriving the prefix sums and evaluating all its split score are both $\mathcal{O}(s)$. Consequently, the overall time complexity of node construction and split installation is $\mathcal{O}(np + ps)$, which is significantly cheaper than conventional $\mathcal{O}(pn \log n)$ for RFs and $\mathcal{O}(nps)$ for ERTs.

Sparser trees. Following Wang et al. (2025), Sparse Forest replaces bootstrap resampling with tree subsampling: each sample belongs to exactly $k = \lceil qT \rceil$ trees (Fig. 1 bottom). A sample modification request therefore touches only k trees, and since each tree holds qN samples on average, the tree that the requests visits are $1/q$ times smaller than tree built by bootstrapping. Therefore, tree subsampling reduces workload by $\mathcal{O}(1/q^2)$, which is 25 times as Sparse Forest adopts $q = 0.2$.

3.2 In-Place Subtree Repair

Structural sparsity reduces the workload of maintaining a *fixed* structure, but an installed split replacement *alters* the structure. Rebuilding the affected subtree is sufficient to restore consistency, but does not exploit the state that remains useful. We found that when an installed split changes, most samples typically routes to the same side, and a large portion of node's descendants and their node statistics still remains valid and reusable. Sparse Forest therefore treats a installed split change as *membership changes* between the two existing children and repairs the affected subtree by only migrating the samples that switch sides.

In-place repair and migration. Suppose the installed split at node u must change from $\mathbf{s}_u$ to $\mathbf{s}'_u$. As a consequence, the samples routed to the children change from $L = D_u^{\mathrm{L}}(\mathbf{s}_u)$, $R = D_u^{\mathrm{R}}(\mathbf{s}_u)$ to $L' = D_u^{\mathrm{L}}(\mathbf{s}'_u)$, $R' = D_u^{\mathrm{R}}(\mathbf{s}'_u)$. Then, for convenience, let A partition D_u by the two splits jointly: $A_{00} \triangleq L \cap L'$, $A_{01} \triangleq L \cap R'$, $A_{10} \triangleq R \cap L'$, and $A_{11} \triangleq R \cap R'$, so that $L = A_{00} \cup A_{01}$ and $R = A_{10} \cup A_{11}$. For the targets (L' and R'), they can be obtained by migrating A_{01} from L to R' and A_{10} from R to L', since $L' = (L \setminus A_{01}) \cup A_{10}$ and $R' = (R \setminus A_{10}) \cup A_{01}$.

Moreover, which child is *left* or *right* is logical, so the two sides of the new split $\mathbf{s}'_u = \langle \alpha', \theta' \rangle$ are chosen by either: *keeping* current selection where left subtree recieve L', and right receive R'; or *swapping* the children so that we assign R' to the left and L' to the right, and alter the "$<$" in split

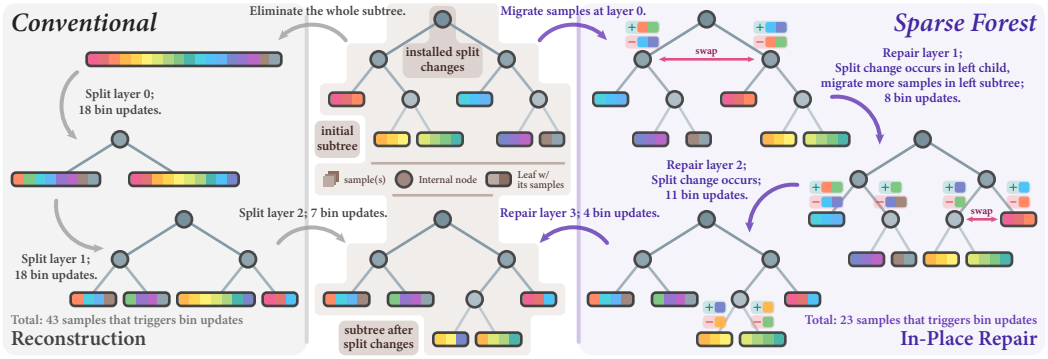


Figure 2: ***Reconstruction Sparsity***: An example of how in-place repair fixes the subtree. **Middle**: the substree in its initial state (top), and its finalized state after the split changes (bottom). **Left**: The conventional way reconstructs the whole subtree when installed split changes. **Right**: Sparse Forest repairs the subtree by recursively (1) identifies the membership difference before-and-after split change; (2) swaps subtree when necessary, and (3) migrates between subtrees and send to children.

condition to "$\geq$". When the decision is to *swap*, the target (R' to the left and L' to the right) obtained by migrating A_{00} from L to L' and A_{11} from R to R'. Therefore, node u migrates $|A_{01}| + |A_{10}|$ samples in *kept* scenario, and $|A_{00}|+|A_{11}|$ samples in *swapped* scenario. Since reversing the subtree is logically $\mathcal{O}(1)$, we choose whichever option migrates fewer samples (with $n = |D_u|$):

$$m \triangleq \min\{|A_{01}| + |A_{10}|, |A_{00}| + |A_{11}|\} \leq \frac{1}{2}(|A_{01}| + |A_{10}| + |A_{00}| + |A_{11}|) = \frac{n}{2}$$

This bound is for worst case scenario, and the migration population m is smaller in practice. Across our experiments, $m \approx 0.37n$ on average when swapping is not allowed, and $m \approx 0.26n$ if swapped when necessary. After the decision, each child receives the migrated sample set, updates its node statistics, and checks its stopping rule and if the installed split needs a change. If its split remains admissible (discuss in Section 3.3), the existing structure is retained and only the membership changes continue downward. Otherwise, the child changes its installed split and invokes the same repair procedure recursively (Fig. 2 right). The node statistics update in children is simply a batch sample addition and deletion on both children, which is m additions and m deletions requiring $\mathcal{O}(mp+ps)$ on each child. Identifying which samples need to be migrated needs an extra $\mathcal{O}(n)$ scan, so the total time complexity is $\mathcal{O}(n+mp+ps)$ in one layer, which is cheaper than reconstruction's $\mathcal{O}(np+ps)$. More empirical analysis of repair and migration can be found in Appendix D.1.

3.3 Suboptimal Split Retention

In-place subtree repair reduces the work required by installed split change, and we next reduce the frequency of such changes. Under the strict optimality requirement ($\varepsilon = 0$), every sample modification request reaching a node u must be reflected in u's node statistics and the optimal split will be re-identified, at a cost of $\mathcal{O}(ps)$ per request for every node on the path. What is worse, whenever the optimal split changes, which is frequent when several candidates are close to the optimal score, the installed split must follow the change, triggering a repair. Both costs stem from insisting that the installed split *equal* the optimal split at all times, so instead, Sparse Forest relaxes the requirement to (ε, φ)-suboptimality (default $\varphi = \frac{1}{2}$) for decoupling installation from optimality.

For any node u, let $D_u^{(t)}$ be the set of samples routed to u at time step t, and "(t)" in superscript denotes time step. To hold (ε, φ)-suboptimality for all $t > 0$, the following condition must be met:

$$\forall t > 0: \quad g_u^{(t)} \triangleq \mathcal{I}\big(D_u^{(t)}, \hat{\mathbf{s}}_u^{(t)}\big) - \min\big\{\mathcal{I}\big(D_u^{(t)}, \mathbf{s}\big) \mid \mathbf{s} \in C_u^{(t)}\big\} \leq \varepsilon \big|D_u^{(t)}\big|^{-\varphi}. \tag{2}$$

Here, $g_u^{(t)}$ denotes the gap between the optimal and installed split, and an installed split is replaced by the current optimal only when it violates Eq. 2, which resets the gap to zero.

However, verifying Eq. 2 naïvely would still cost $\mathcal{O}(ps)$ per request, since $g_u^{(t)}$ involves evaluating score for all candidates and obtaining the minimum. The following observation make it cheaper, and brings about two positive consequences:

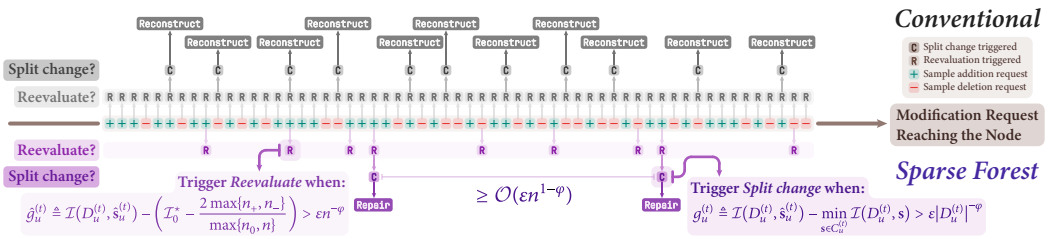


Figure 3: ***Split-Change Sparsity***: Sparse Forest (**Bottom**) trigger far less *Reevaluation* and *Split change*, and use cheaper *Repair* instead of the expensive *Reconstruction* used by convention (**Top**).

Lemma 1: Score Changes under Modification

Let D' obtained from a nonempty dataset D by a sequence of n_+ insertions and n_- deletions, with $n = |D| > 0$ and $n' = |D'| > 0$. For any fixed split $\mathbf{s}$ evaluated under Gini index $\mathcal{I}(\cdot,\cdot)$,

$$\left|\mathcal{I}(D,\mathbf{s}) - \mathcal{I}(D',\mathbf{s})\right| \leq \frac{2\max\{n_+,n_-\}}{\max\{n,n'\}}. \tag{3}$$

Less Frequent Reevaluation. By Eq. 3, a single sample insertion or deletion changes any split score by at most $2/n$, and $g_u^{(t)}$ increases at most $4/n = \mathcal{O}(1/n)$ whenever a new modification request reaches u (in the worst case, the installed split worsen by $2/n$ while the optimal improves by $2/n$). Since we set the installed split to the optimal when Eq. 2 fails, it resets $g_u^{(t)} = 0$, so it needs at least $\varepsilon n^{-\varphi}/(4/n) = \mathcal{O}(\varepsilon n^{1-\varphi})$ modification requests to invalidate Eq. 2 and initiate a new replacement. This inspires the *batch reevaluation* mechanism that maintains the node statistics and checks suboptimality validation after $\mathcal{O}(\varepsilon n^{1-\varphi})$ modification requests since last split replacement. Therefore, (1) split replacements could only happen during reevaluation; (2) every modification request reaches u no longer neccesitates reevaluation as before; and (3) between consecutive reevaluations, modification requests are cached and used to batch update the node statistics during reevaluation.

An $\mathcal{O}(1)$ gap estimation. With batch reevaluation, an $\mathcal{O}(1)$ approximation to Eq. 2 is possible:

$$\forall\, t > 0: \quad \hat{g}_u^{(t)} \triangleq \mathcal{I}\big(D_u^{(t)}, \hat{\mathbf{s}}_u^{(t)}\big) - \left(\mathcal{I}_0^\star - \frac{2\max\{n_+,n_-\}}{\max\{n_0,n\}}\right) \leq \varepsilon n^{-\varphi}, \tag{4}$$

with $\mathcal{I}_0^\star$ being the optimal split score obtained in last reevaluation, n_0 being the number of samples in u in last reevaluation, and n_+, n_- being the number of sample addition/deletion requests that reaches u since last reevalution, and $n = n_0 + n_+ - n_-$. Therefore, whenever a sample modification request reaches u, we use Eq. 4 to check if reevaluation is necessary in $\mathcal{O}(1)$.

Consequently, between reevaluations, a modification request cost only $\mathcal{O}(1)$ at u for caching and verifying Eq. 4. As discussed, modification requests are batched and applied during reevaluation, which requires an $\mathcal{O}(n_\pm \cdot p + ps)$ node statistics update (Section 3.2), where $n_\pm \triangleq n_+ + n_-$ denotes the number of modification requests between two consecutive reevaluations. Therefore, the amortized cost for node u processing such modification request is $\mathcal{O}(p + \frac{ps}{\varepsilon n^{1-\varphi}})$. More time complexity analysis is available at Appendix B.3, discussion of reevaluation frequency is at Appendix D.2.

4 Experiments

We evaluate Sparse Forest (SF) along three criteria: *predictive performance*, *exactness*, and *efficiency*. The objective is to minimize the cost of processing the online (un)learning request stream while preserving predictive performance and satisfying the chosen exactness requirement. Our evaluation covers online (un)learning request stream AUC, request stream process time, KL divergence compare to retrained model on nine binary classification datasets, and experiments on multi-class classification and regression can be found in Appendix C.5. Dataset descriptions, model configurations, request streams, and measurement procedures are given in Appendix C, with more results in the remaining appendices.

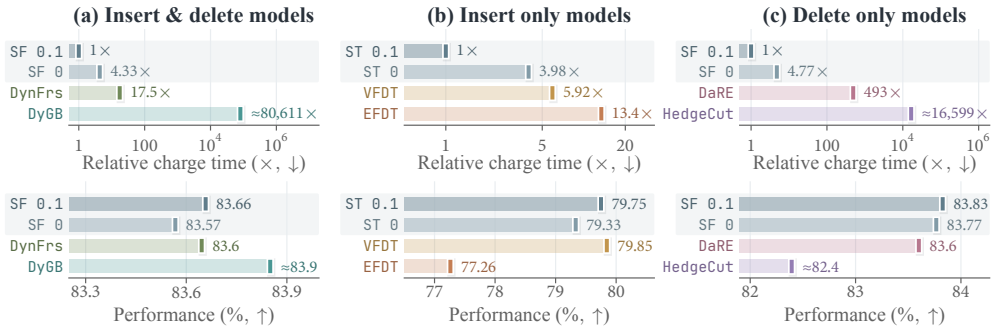


Figure 4: Overall maintenance time (top, ↓) and query performance during stream (bottom, ↑) of all (**Left**: insert & delete, **Middle**: insert-only, **Right**: delete-only) models. DynFrs timing includes runs affected by shared-server contention; DaRE time and performance include estimates.

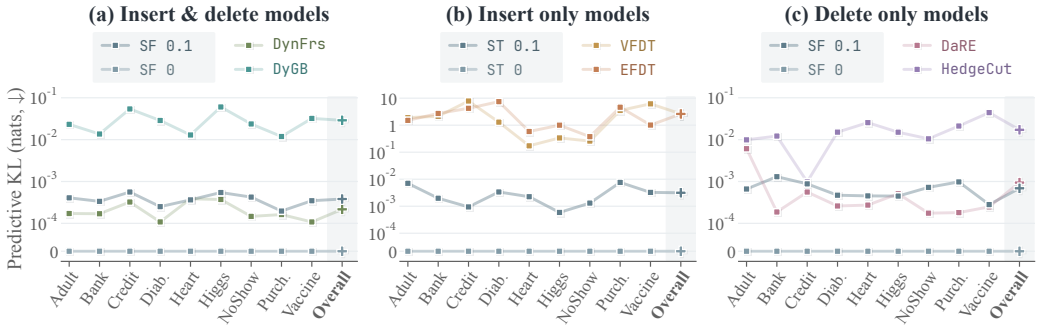


Figure 5: Divergence (↓) between maintained model and retrained model, which measures the exactness of each model. Sparse Forest with $\varepsilon = 0$ achieves total exactness in all tests.

Predictive performance and online efficiency. Fig. 4 compares predictive performance with request stream processing time. In the online learning and unlearning setting, both SF ($\varepsilon = 0$) and ($\varepsilon = \frac{1}{10}, \varphi = \frac{1}{2}$) require substantially less processing time than DynFrs (Wang et al., 2025), while their predictive performance remains close. DyGB have higher predicitive performance as it is in different family of tree ensembles (GBDTs), and it is built for batch request processing (Appendix C.4, Fig 12). For models only capabile of online learning, Sparse Tree (ST) with $\varepsilon = 0.1$ performs comparably to VFDT and better than EFDT (Manapragada et al., 2018), and processes the stream $5.9\times$ and $13.4\times$ faster, respectively. For deletion-only models, SFs are significantly faster than DaRE (Brophy & Lowd, 2021) and HedgeCut (Schelter et al., 2021), with broadly comparable predictive performance. Across all three settings, the SF $\varepsilon = 0.1$ further reduce processing time relative to $\varepsilon = 0$ by about $4\times$ with little change in predictive performance.

Exactness. Fig. 5 therefore compares each maintained model with its own training algorithm applied afresh to the updated dataset, $\mathsf{M}_{\text{re}} = \mathcal{A}(D_t)$. With the randomness fixed, SF and ST at $\varepsilon = 0$ are bit-identical (with zero KL divergence) to their retrained models at every evaluation. The equality checks for $\varepsilon = 0$ cover predictions and model state, including node sample sets, installed splits, and leaf statistics, which provides empirical support for SF's total exactness. With suboptimal split retention, predictive divergence remains small but is no longer zero: the model can preserve predictive performance without being identical to retraining. Notice that the plotted KL divergence is solely a proxy to distributional exactness. Appendix C.4 specifies the retraining references, states how the KL divergence is measured, and reports the full equality and divergence checks.

Effect of suboptimal split retention. Fig. 6 examines how the (ε, φ)-suboptimality condition affects all three criteria while holding the remaining configuration fixed. Among the positive-tolerance settings, increasing ε or decreasing φ (loosen the bound) generally reduces processing time and increases divergence from retraining, whereas AUC changes little over the tested range. Stricter conditions, on the other hand, retain closer agreement with retraining but require more maintenance work. Thus, on these datasets, the principal observed tradeoff introduced by split retention is between ef-

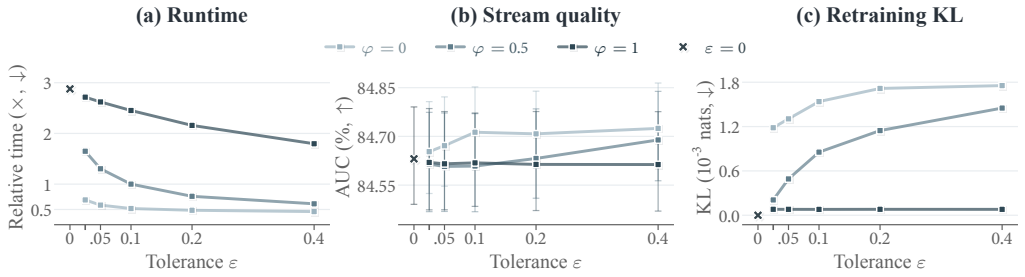


Figure 6: Diverse behaviors of different (ε, φ)-suboptimality conditions in efficiency (**Left**, ↓), predictive performance (**Middle**, AUC, ↑), and exactness (**Right**, KL divergence between maintained and retrained model, ↓). Stricter conditions use darker colors.

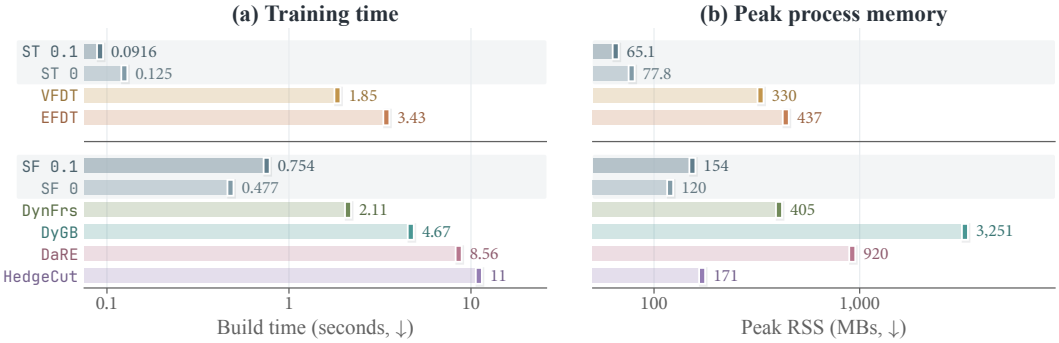


Figure 7: Training time (**Left**, ↓) and peak memory usage (**Right**, ↓) of all models across all datasets.

ficiency and agreement with retraining, rather than between efficiency and predictive performance, as there are no significant scaling effect in stream AUC.

Training time and memory. Fig. 7 complements the stream comparisons with initial training time and peak process memory. On the dataset subsets shown, SFs and STs train faster and use less peak memory than their respective baselines, although these aggregate savings are not universal across individual datasets. Importantly, training time does not determine maintenance efficiency: exact SF builds faster than SF $\varepsilon = 0.1$, yet takes longer to process the streams in Figure 4. Online (un)learning must therefore be evaluated on the request stream itself, together with predictive performance and the required exactness, rather than on training cost alone. Interestingly, SFs and STs use significantly less memory compared to other models, which use 62% less memory compare to the second fastest model (DynFrs).

5 Conclusion

In this study, we presented *Sparse Forest* (SF), an efficient tree esemble online learning and unlearning framework with configuration exactness. SF's three complementary forms of sparsity allievate distinct sources of maintenance cost: *structural sparsity* limits the structure and statistics that must be maintained; *reconstruction sparsity* reuses existing descendants through in-place subtree repair; and *split-change sparsity* reduces reevaluations and split replacements through suboptimal split retention. Together, these mechanisms reduce the overall cost of maintaining the tree ensemble, how much work each repair requires, and how often repairs occur. Additionally, SF makes the choice between total exactness and suboptimal split retention explicit. Under a fixed training specification and shared randomness, its totally exact configuration ($\varepsilon = 0$) maintains a model bit-identical to retraining on the current dataset. Positive tolerance in the (ε, φ)-suboptimality condition relinquishes this equivalence in exchange for further efficiency gains. Across the evaluated workloads, SF substantially reduces request-processing time by being $16.9\times$ faster than the best existing models on all datasets and $81.8\times$ faster on the largest dataset while retaining competitive predictive performance and using only 38% of memory. Furthermore, its exact configuration matches retraining bit for bit in all completed equality checks, and is still several times faster than existing models.

AI USE STATEMENT

Generative AI assisted literature discovery, manuscript drafting and editing, mathematical checks, and scripts for result extraction and visualization. A strong model is also be used for providing manuscript critiques. AI agents also be used to ensure all reported numerical results were computed from experiment records rather than generated as text.

ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. In this study, no human subjects or animal experimentation was involved. All datasets used were sourced in compliance with relevant usage guidelines, ensuring no violation of privacy. We have taken care to avoid any biases or discriminatory outcomes in our research process. No personally identifiable information was used, and no experiments were conducted that could raise privacy or security concerns. We are committed to maintaining transparency and integrity throughout the research process.

REPRODUCIBILITY STATEMENT

Code We provide code to help understand this work, and is publicly available at: https://anonymous.4open.science/r/Sparse-Forest-M520/.

Datasets All datasets are either included in the repo, or a description for how to download and preprocess the dataset is provided. All datasets are public and raise no ethical concerns.

Hyperparameters All parameters of our proposed framework are search using a fixed protocol provided in the codebase.

Environment Details of our experimental setups are provided in Appendix C.

Random Seed we do not set a random seed specifically for all random behavior, with the random seed determined PyTorch.

REFERENCES

Stefan Aeberhard and M. Forina. Wine. UCI Machine Learning Repository, 1992. URL https://doi.org/10.24432/C5PC7J.

E. Alpaydin and C. Kaynak. Optical recognition of handwritten digits. UCI Machine Learning Repository, 1998. URL https://doi.org/10.24432/C50P49.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5:4308, 2014. doi: 10.1038/ncomms5308.

Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. URL https://doi.org/10.24432/C5XW20.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 141–159. IEEE, 2021. doi: 10.1109/SP40001.2021.00019. URL https://arxiv.org/abs/1912.03817.

Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. doi: 10.1023/A:1010933404324.

Marco Bressan and Mauro Sozio. Fully-dynamic approximate decision trees with worst-case update time guarantees. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 4517–4541, 2024. URL https://proceedings.mlr.press/v235/bressan24a.html.

Marco Bressan, Gabriel Damay, and Mauro Sozio. Fully-dynamic decision trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023. URL https://arxiv.org/abs/2212.00778.

Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1092–1104, 2021. URL https://proceedings.mlr.press/v139/brophy21a.html.

Ramon Casanova, Santiago Saldana, Emily Y. Chew, Ronald P. Danis, Craig M. Greven, and Walter T. Ambrosius. Application of random forests methods to diabetic retinopathy classification analyses. *PLOS ONE*, 9(6):e98587, 2014. doi: 10.1371/journal.pone.0098587.

John Clore, Krzysztof Cios, Jon DeShazo, and Beata Strack. Diabetes 130-US hospitals for years 1999–2008. UCI Machine Learning Repository, 2014. URL https://doi.org/10.24432/C5230J.

Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*. IEEE, 2015. doi: 10.1109/SSCI.2015.33. URL https://ieeexplore.ieee.org/document/7376606/.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 71–80. ACM, 2000. URL https://homes.cs.washington.edu/~pedrod/papers/kdd00.pdf.

DrivenData. Flu shot learning: Predict H1N1 and seasonal flu vaccines, 2019. URL https://www.drivendata.org/competitions/66/flu-shot-learning/page/211/.

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004. doi: 10.1214/009053604000000067.

R. A. Fisher. Iris. UCI Machine Learning Repository, 1936. URL https://doi.org/10.24432/C56C76.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63:3–42, 2006. doi: 10.1007/s10994-006-6226-1.

Abderahmane Hamdouchi and Ali Idri. Evaluating the performance of TinyML singular and ensemble techniques for intrusion detection in IoT networks. *Microprocessors and Microsystems*, 117: 105172, 2025. doi: 10.1016/j.micpro.2025.105172.

Joni Hoppen and Aquarela Analytics. Medical appointment no shows. Kaggle, 2016. URL https://www.kaggle.com/datasets/joniarroba/noshowappointments.

Jochen Kruppa, Alexandra Schwarz, Gerhard Arminger, and Andreas Ziegler. Consumer credit risk: Individual probability estimates using machine learning. *Expert Systems with Applications*, 40(13):5125–5131, 2013. doi: 10.1016/j.eswa.2013.03.019.

Huawei Lin, Jun Woo Chung, Yingjie Lao, and Weijie Zhao. Online gradient boosting decision tree: In-place updates for efficient adding/deleting data. arXiv preprint arXiv:2502.01634, 2025. URL https://arxiv.org/abs/2502.01634.

Afonso Lourenço, João Rodrigo, João Gama, and Goreti Marreiros. DFDT: Dynamic fast decision tree for IoT data stream mining on edge devices. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(29):24053–24060, 2026. doi: 10.1609/aaai.v40i29.39583. URL https://ojs.aaai.org/index.php/AAAI/article/view/39583.

Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi. Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018. URL https://arxiv.org/abs/1802.08780.

Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014a. doi: 10.1016/j.dss.2014.03.001.

Sérgio Moro, Paulo Rita, and Paulo Cortez. Bank marketing. UCI Machine Learning Repository, 2014b. URL https://doi.org/10.24432/C5K306.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011. URL https://jmlr.org/papers/v12/pedregosa11a.html.

R. Quinlan. Auto MPG. UCI Machine Learning Repository, 1993. URL https://doi.org/10.24432/C5859H.

C. Sakar and Yomi Kastro. Online shoppers purchasing intention dataset. UCI Machine Learning Repository, 2018. URL https://doi.org/10.24432/C5F88Q.

Sebastian Schelter, Stefan Grafberger, and Ted Dunning. HedgeCut: Maintaining randomised trees for low-latency machine unlearning. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 1545–1557, 2021. doi: 10.1145/3448016.3457239. URL https://deem.berlin/pdf/rdm235.pdf.

Beata Strack, Jonathan P. DeShazo, Chris Gennings, Juan L. Olmo, Sebastian Ventura, Krzysztof J. Cios, and John N. Clore. Impact of HbA1c measurement on hospital readmission rates: Analysis of 70,000 clinical database patient records. *BioMed Research International*, 2014:781670, 2014. doi: 10.1155/2014/781670.

Svetlana Ulianova. Cardiovascular disease dataset. Kaggle, 2018. URL https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset.

Shurong Wang, Zhuoyang Shen, Xinbao Qiao, Tongning Zhang, and Meng Zhang. DynFrs: An efficient framework for machine unlearning in random forest. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.01588.

Heng-Ru Zhang and Fan Min. Three-way recommender systems based on random forests. *Knowledge-Based Systems*, 91:275–286, 2016. doi: 10.1016/j.knosys.2015.06.019.

A PSEUDOCODE

We first describe how the three sparsity mechanisms fit into one update procedure. The pseudocode works at the level of node statistics and sample membership; it does not prescribe a particular container or memory layout. In particular, writing D_u does not require a separate copy of every sample at every node.

Node state. A node maintains its installed split $\hat{\mathbf{s}}_u$, the statistics needed to evaluate that split, and the histograms for its candidate attributes. Between reevaluations, changes to the candidate histograms may be buffered in a list Q_u of signed sample updates. The installed split's class counts are kept current, so its Gini score can still be evaluated without updating all candidate histograms. The node also records n_0 and $\mathcal{I}_0^\star$ from its last reevaluation, together with the numbers n_+ and n_- of subsequent insertions and deletions. These are the quantities used by the gap bound in Section D.2.

Repeated copies of a record have distinct occurrence identifiers. An insertion introduces an occurrence, and a deletion removes an occurrence that is currently present. This makes both tree membership and the set differences used during repair unambiguous.

Construction rules. The routines below use the same candidate-generation, stopping, and prediction rules as the training algorithm $\mathcal{A}$. In the totally exact configuration, their randomness and tie-breaking must also follow the fixed specification in Section B.2. Whenever a candidate set, feature range, logical node position, or split-eligibility rule changes, any statistics or gap bound that depend on the old specification must be refreshed before use. A stopping decision that cannot be certified from the maintained state also requires reevaluation.

Algorithm 1 Construct a subtree

function BUILD(D_u, u)
  Initialize the node's sample count and class counts from D_u
  if a stopping rule applies without a split search **then**
    return a leaf with the prescribed prediction statistics
  end if
  Construct C_u using the prescribed attributes and threshold grids
  Accumulate the bin counts and compute their prefix sums
  Evaluate eligible candidates and apply the remaining stopping rules
  if the node must be a leaf **then**
    return a leaf with the prescribed prediction statistics
  end if
  $\hat{\mathbf{s}}_u \leftarrow$ the best eligible candidate, using the prescribed tie rule
  Record $n_0 \leftarrow |D_u|$, $\mathcal{I}_0^\star \leftarrow \mathcal{I}(D_u, \hat{\mathbf{s}}_u)$
  Set $Q_u \leftarrow \varnothing$ and $n_+ \leftarrow n_- \leftarrow 0$
  Build the two children from $D_u^{\mathrm{L}}(\hat{\mathbf{s}}_u)$ and $D_u^{\mathrm{R}}(\hat{\mathbf{s}}_u)$
  return u
end function

Updating a node. Algorithm 2 receives the samples added to and removed from a node. These may come from an external request or from a parent's repair. The same procedure handles both: a sample moving into a child is an insertion at that child, and a sample moving out is a deletion. Candidate-statistics updates may be batched, but membership changes and prediction statistics must not be silently dropped.

Here $\hat{g}_u$ is evaluated only when its reference candidate set and construction rules remain valid. The flag F is not assumed to be free: any work needed to detect or process such changes belongs to maintenance cost. At $\varepsilon = 0$, the best candidate is installed even when the old split has the same score but loses the prescribed tie-break.

An exchanged child is a reused storage object, not permission to change the reference learner. If its new logical position requires different candidates or randomness, that state must be regenerated. Algorithm 3 permits an exchange only with this requirement enforced. For a fixed parent population, its membership calculation reduces to the $A_{00}, A_{01}, A_{10}, A_{11}$ construction in the main text.

Algorithm 2 Update a node and its affected descendants

function UPDATE(u, D^+, D^-)
  Remember the old installed split, child memberships, and construction context
  $D_u \leftarrow (D_u \setminus D^-) \cup D^+$
  Update the node's class counts and, if present, its installed-split counts
  if a stopping rule already certifies that u must be a leaf **then**
    Discard the descendants and return a leaf with current statistics
  end if
  if u has no installed split **then**
    return BUILD(D_u, u)
  end if
  Append the signed changes to Q_u; update n_+ and n_-
  $F \leftarrow$ whether the construction, eligibility, or stopping rules require refresh
  if $\varepsilon = 0$ or F or $\hat{g}_u > \varepsilon |D_u|^{-\varphi}$ **then**
    Apply Q_u to the histograms, or reconstruct them if their specification changed
    Evaluate eligible candidates and apply the stopping rules
    if the node must be a leaf **then**
      Discard the descendants and return a leaf with current statistics
    end if
    $\mathbf{s}_u^\star \leftarrow$ the best eligible candidate with the prescribed tie rule
    if $\varepsilon = 0$, or $\hat{\mathbf{s}}_u$ is ineligible, or $\mathcal{I}(D_u, \hat{\mathbf{s}}_u) - \mathcal{I}(D_u, \mathbf{s}_u^\star) > \varepsilon |D_u|^{-\varphi}$ **then**
      $\hat{\mathbf{s}}_u \leftarrow \mathbf{s}_u^\star$
    end if
    Refresh the installed-split counts
    $n_0 \leftarrow |D_u|, \mathcal{I}_0^\star \leftarrow \mathcal{I}(D_u, \mathbf{s}_u^\star)$
    Clear Q_u and reset n_+, n_- to zero
  end if
  REPAIR(u, D^+, D^-, old state)
  return u
end function

Algorithm 3 Repair child memberships after a node update

function REPAIR(u, D^+, D^-, old state)
  if the installed split and child construction contexts are unchanged **then**
    Route D^+ and D^- through the installed split
    Replace each affected child by the result of UPDATE on its changes
    return
  end if
  Form the target memberships $L' = D_u^{\mathrm{L}}(\hat{\mathbf{s}}_u)$ and $R' = D_u^{\mathrm{R}}(\hat{\mathbf{s}}_u)$
  Consider keeping or exchanging the two stored children
  Choose a permitted assignment minimizing the total membership changes
  for each logical child position with target membership S **do**
    if no child is available for reuse **then**
      Install BUILD(S, child position) at this child position
    else
      Let v be the assigned child, with its previous membership D_v
      Refresh its logical construction context if necessary
      if membership or construction context changed **then**
        $v \leftarrow$ UPDATE($v, S \setminus D_v, D_v \setminus S$)
      end if
      Assign v to this logical child position
    end if
  end for
end function

Forest requests and deferred work. Training calls BUILD once for each tree on the occurrences assigned to that tree. An insertion or deletion replaces each affected root with the result of UPDATE, visiting only the trees selected by the sample's tree-membership rule. A query uses the same leaf predictions and ensemble aggregation as $\mathcal{A}$.

The pseudocode completes logical subtree repair before returning an update. A lazy implementation may change this schedule, but it must resolve any pending work before using the affected state for prediction or comparison. Equivalence of a fully materialized model does not, by itself, imply that two unfinished queue or cache representations are bit-identical. The exactness statement below therefore makes the compared model state explicit.

B PROOF

B.1 LIGHT-WEIGHT BIN UPDATE EQUIVALENCE

The child sample counts and class counts determine the Gini or entropy split score:

$$\mathcal{I}(D, \mathbf{s}_j) = \sum_{\sigma \in \{\mathrm{L},\mathrm{R}\}} \frac{n^\sigma}{n} \left[1 - \sum_{y \in \mathcal{Y}} \left(\frac{n_y^\sigma}{n^\sigma} \right)^2 \right] \qquad \text{or} \qquad \mathcal{I}(D, \mathbf{s}_j) = \sum_{\sigma \in \{\mathrm{L},\mathrm{R}\}} \frac{n^\sigma}{n} \left[- \sum_{y \in \mathcal{Y}} \frac{n_y^\sigma}{n^\sigma} \log \frac{n_y^\sigma}{n^\sigma} \right] \quad (5)$$

Fix a node and an attribute, and suppress their indices. Let $l < r$, $\Delta = (r - l)/(s + 1)$, and $\theta_j = l + j\Delta$ for $j = 1, \ldots, s$. The bins must agree with the routing rule $x_\alpha < \theta_j$. In exact arithmetic, a definition that also handles the endpoints is

$$b(\mathbf{x}) = \min \left\{ s, \max \left\{ 0, \left\lfloor \frac{x_\alpha - l}{\Delta} \right\rfloor \right\} \right\}. \quad (6)$$

In particular, $x_\alpha = r$ belongs to bin s, not bin $s + 1$. Constant attributes have no nontrivial grid and are handled by the construction rule rather than by dividing by zero.

Proposition B.1 (Bin equivalence)**.** *For every candidate $\mathbf{s}_j = \langle \alpha, \theta_j \rangle$, the child counts obtained from histogram prefix sums equal those obtained by routing the samples individually. Signed histogram updates preserve this equality after every valid sequence of insertions and deletions.*

Proof. For $1 \le j \le s$, the bin definition gives

$$b(\mathbf{x}) < j \quad \Longleftrightarrow \quad x_\alpha < \theta_j.$$

Therefore, writing c_β for a bin count,

$$\sum_{\beta=0}^{j-1} c_\beta = \sum_{\langle \mathbf{x}_i, y_i \rangle \in D} \mathbf{1}\{b(\mathbf{x}_i) < j\} = \sum_{\langle \mathbf{x}_i, y_i \rangle \in D} \mathbf{1}\{x_{i,\alpha} < \theta_j\} = |D^{\mathrm{L}}(\mathbf{s}_j)|.$$

Restricting the same identity to class y gives

$$n_y^{\mathrm{L}} = \sum_{\beta=0}^{j-1} c_{y,\beta}, \qquad n^{\mathrm{R}} = n - n^{\mathrm{L}}, \qquad n_y^{\mathrm{R}} = n_y - n_y^{\mathrm{L}}.$$

Both Gini and entropy scores are functions of these counts, so the two ways of evaluating a fixed candidate give the same score.

An insertion adds one to exactly its total bin count and its class-specific bin count; a deletion subtracts one from those same counts. Thus every bin continues to equal the sum of its sample indicators. Induction over the update sequence completes the proof. □

Empty children contribute zero to the weighted impurity; entropy additionally uses $0 \log 0 = 0$. Whether an empty-child split is eligible for installation is a separate construction rule and must be shared by training and maintenance.

For stored machine thresholds, the exact bin definition is $b(\mathbf{x}) = |\{j : x_\alpha \ge \theta_j\}|$, with thresholds in nondecreasing order. It guarantees $b(\mathbf{x}) < j$ if and only if $x_\alpha < \theta_j$, including equality at a threshold. An arithmetic shortcut for binning must reproduce this index; an uncorrected floating-point floor is insufficient. Exact integer counts without overflow and the same score-evaluation program then give bit-identical scores. This equivalence concerns the prescribed grid, not its approximation quality relative to unrestricted thresholds.

B.2 TOTAL EXACTNESS

For the count-based classification learner, we prove equality of both the node statistics and the installed splits with fresh training on the current active data. Zero tolerance is used together with the following construction rules; setting a numerical tolerance to zero without these rules is insufficient.

Fixed encoding and occurrence assignment. Fix the feature encoding Q, all learner hyperparameters, and a random seed ω. Preprocessing is fitted once and shared with fresh training. We regard D_t as a multiset: repeated samples retain their multiplicities, and a deletion removes one live occurrence. A deterministic, sample-keyed function $A_\omega(Q(\mathbf{x}), y)$ selects the k distinct trees receiving a sample. The function may ignore y, as in the native binary implementation; it is fixed for both procedures and does not use insertion order or transient storage identifiers. Hence each tree has exactly the same root population under maintenance and retraining. Occurrence identifiers distinguish copies for deletion, but are not additional randomness. When comparing bytes, populations are serialized as sorted encoded sample multisets, independently of storage order.

Ancestor-range and tree-path encoded randomness. Write a logical node as $u = (\tau, p)$, where τ is its tree index and $p \in \{0,1\}^{\le h}$ is its complete left/right path from the root. We use *tree-path encoded randomness*: the node's random stream is a deterministic function of (ω, τ, p) and its draw indices. It is independent of allocation addresses, update order, query order, and the number of previous repairs. Reconstructing the same logical node restarts that stream.

For the ancestor-range formulation, we use the fixed encoded feature domain intersected with the routing constraints of the current ancestors. With the predicate $x_\alpha < \theta$, the endpoints are

$$l_{u,\alpha} = \max\big(\{l_{\mathrm{root},\alpha}\} \cup \{\theta_v : v \prec u,\ \alpha_v = \alpha,\ p \text{ follows } v\text{'s right child}\}\big),$$

$$r_{u,\alpha} = \min\big(\{r_{\mathrm{root},\alpha}\} \cup \{\theta_v : v \prec u,\ \alpha_v = \alpha,\ p \text{ follows } v\text{'s left child}\}\big).$$

Here $v \prec u$ denotes a strict ancestor, and endpoint inclusion follows the routing predicate. These bounds depend on installed ancestors, not historical sample extrema. Denote their collection by R_u. The *ordered* candidate bank is

$$C_u = (\mathbf{s}_{u,1}, \ldots, \mathbf{s}_{u,m_u}) = \mathrm{Generate}(Q, R_u, \omega, \tau, p). \tag{7}$$

Attributes are drawn from a canonically ordered, encoding-fixed pool; even threshold grids, rounding, and candidate order follow one fixed construction program. Degenerate ranges are handled deterministically. A currently one-sided candidate is marked ineligible, rather than replaced by a history-dependent redraw. Thus identical ancestor splits imply identical bounds, random choices, candidate order, and threshold bits. If an ancestor split changes, dependent grids, histograms, and certificates are refreshed recursively, even at descendants whose memberships do not change. A stored child moved to a different logical path must likewise regenerate its path-dependent construction state; keeping its old random choices is invalid.

Arithmetic, ties, and the compared state. Counts and signed count updates use exact integers without overflow, including intermediate sums. Both procedures use the same machine types, arithmetic environment, score-evaluation order, stopping predicates, leaf prediction rules, and ensemble summation order. No equivalence between differently rounded score formulas is assumed. Among eligible candidates, ties are resolved by the first candidate in the canonical bank order. There is no incumbent preference, positive margin floor, or adaptive redraw at zero tolerance.

The compared model consists of the canonically represented logical populations, candidate descriptors and sufficient statistics, installed splits, tree structure, and leaf statistics. Allocation indices, padding, spare capacity, stale score caches, and pending-work representations are excluded. Lazy execution may defer an operation only while preserving its signed multiplicity and logical destination: each change is either applied or pending, never lost or duplicated. The compared model is the result of complete materialization; a query must resolve all dependencies on its traversal before using them.

Lemma B.2 (Bit-identical node statistics)**.** *At a logical node with the same population and candidate grid, signed bin maintenance and fresh accumulation yield bit-identical sample counts, class counts, bin counts, and candidate child counts.*

Proof. For a fixed attribute α and its current bin map $b_{u,\alpha}$, write

$$H_{u,\alpha,\beta,y}(S) = \sum_{\langle \mathbf{x}_i, y_i \rangle \in S} \mathbf{1}\{b_{u,\alpha}(Q(\mathbf{x}_i)) = \beta,\ y_i = y\}.$$

The sum counts occurrences, including repeated copies. Suppose a reused node changes from population B to target population S. Let $B^- = B \setminus S$ and $B^+ = S \setminus B$, with multiset differences (or equivalent differences of occurrence identifiers). Then, entry by entry,

$$H(B) - H(B^-) + H(B^+) = H(S). \tag{8}$$

This is precisely the signed update performed during maintenance. It holds for batches and for migrations as well as external insertions and deletions. The same indicator argument gives n_u and each class count $n_{u,y}$. All additions and subtractions are exact, so batching and delivery order cannot change their final integer representations. If the grid changes, the old histogram is discarded or exactly reconstructed under the new bin map before use; Equation equation 8 never mixes different grids.

By Section B.1, prefix sums give the same candidate left counts as individual routing, and subtraction from node totals gives the same right counts. Thus every sufficient count agrees bit for bit with fresh accumulation. Buffered updates satisfy the same identity after being applied exactly once; a partial physical histogram is not itself the materialized statistic. □

Lemma B.3 (Bit-identical installed split)**.** *If a node's population and ancestor splits agree with fresh training, then its stopping decision and, when internal, its installed split agree bit for bit at* $\varepsilon = 0$.

Proof. Identical ancestors give identical ancestor ranges. Equation equation 7 and the shared path stream therefore give the same ordered candidate descriptors, including threshold bits. Lemma B.2 gives identical sufficient counts. Applying the same stopping and eligibility rules to these counts and the same depth gives identical decisions.

If a split is required, let E_u be the common set of eligible candidate indices and let

$$s_{u,j} = \mathrm{Eval}(\text{candidate } j\text{'s sufficient counts})$$

be the score produced by the prescribed machine program. The inputs and operation sequence coincide, so the score bits coincide. Both procedures select

$$j_u^\star = \min\{j \in E_u : s_{u,j} = \min_{i \in E_u} s_{u,i}\}, \qquad \hat{\mathbf{s}}_u = \mathbf{s}_{u,j_u^\star}. \tag{9}$$

An empty eligible set produces a leaf according to the common stopping rule. Algorithm 2 installs this canonical winner at zero tolerance, including when the old split has an equal score but a later index. Hence the installed attribute, threshold, and candidate index are identical. Merely checking that the incumbent's score gap is zero would not establish this claim: two tied candidates can route different samples. A skipped evaluation is valid only if its certificate proves that the incumbent is still this canonical winner, including the tie rule and stopping decision. □

Theorem B.4 (Total exactness at zero tolerance)**.** *Under the construction and arithmetic rules above, let* $\mathsf{M}_0 = \mathcal{A}(D_0; Q, \omega)$ *and let Sparse Forest process any valid finite stream using Algorithms 2–3 with* $\varepsilon = 0$*. For every request prefix* t*, its materialized logical model satisfies*

$$\mathsf{M}_t = \mathcal{A}(D_t; Q, \omega)$$

bit for bit. Every query also returns the same prediction bits as this fresh learner.

Proof. We induct over requests and, within each update, from the root down to depth h. Initialization is fresh training itself. For an insertion or deletion, the fixed occurrence assignment supplies each tree with the same target root population as retraining on D_t. Unaffected trees retain the same populations and specification.

Consider a logical node whose ancestors have already been made identical to the fresh tree and whose target population is S. A newly constructed node runs the shared builder. For a reused node, repair either updates a compatible histogram by the exact membership differences or reconstructs the bank and histogram for the new path and ancestor ranges. Lemma B.2 gives the fresh statistics, and Lemma B.3 gives the fresh stopping decision and installed split. If a leaf is required, both models remove any descendants and use the same counts and prediction program, including the fixed empty-leaf convention.

Otherwise the identical split partitions S into identical target child populations S_0, S_1. For a stored child with old population B, Algorithm 3 removes $B \setminus S_b$ and adds $S_b \setminus B$, so

$$\big(B \setminus (B \setminus S_b)\big) \uplus (S_b \setminus B) = S_b, \qquad b \in \{0, 1\},$$

where $\uplus$ adds multiplicities. Repair also propagates changed construction context, including when both membership differences are empty. Thus the induction hypotheses hold at each child. Reusing storage, choosing a full rebuild, or exchanging children with complete context refresh changes only the work performed. The finite depth bound terminates the induction and proves equality of every logical node, its statistics, and its split. Canonical tree and population order then give equality of the serialized forest.

A query changes no active population. In a lazy implementation, pending signed changes remain accounted for until applied, and repair proceeds through the ancestors before a child route or leaf statistic is read. The same argument therefore establishes equality along each queried path; complete materialization establishes it throughout the forest. The query reaches the same leaves and uses identical prediction and ensemble arithmetic, hence returns identical bits. This completes the induction over all request prefixes. □

Relation to the native exact implementation. The native binary zero-tolerance engine uses a fixed *full-domain* candidate grid, rather than ancestor-restricted grids. It is covered by the same proof with $R_u = R_{\text{root}}$ for candidate generation. It uses encoded-feature-keyed occurrence, complete tree-path seeds, canonical first-minimum ties, and disables adaptive redraw and child exchange. This is equality with fresh training under that same full-domain specification; it does not assert equality between full-domain and ancestor-range learners. The ancestor-range formulation above requires the stated descendant refresh rule.

The theorem is pointwise in the shared seed. Consequently it also gives distributional exactness when maintenance and retraining use the same seed distribution for a fixed request stream and encoding. It concerns logical model state in a common arithmetic environment, not physical memory images or refitting preprocessing on the surviving raw data. For $\varepsilon > 0$, a different installed split may be retained, so the split lemma and retraining-equality conclusion no longer follow.

B.3 Time Complexity

We first separate ordinary statistics maintenance from structural repair. Unless stated otherwise, the number of classes is fixed, as in the binary-classification experiments. We assume constant-time access to a record and its bin index, and indexed membership updates. Candidate regeneration, a changed stopping decision, and work performed by lazy queries are counted separately when they occur.

Constructing a node and a forest. At a node with n_u samples, accumulating p attribute histograms costs $O(n_u p)$, and scanning their prefixes and candidate scores costs $O(ps)$. Thus construction costs $O(n_u p + ps)$. If the number of classes is c rather than constant, the scan term is $O(psc)$.

Let V be the total number of materialized nodes, h the maximum tree depth, and N the number of active occurrences. Suppose each occurrence is assigned to k trees. Summing node construction costs gives

$$O\left(\sum_u n_u p + Vp(s+1) + T\right) \subseteq O(kNp(h+1) + Vp(s+1) + T)\,. \tag{10}$$

The last bound follows because an occurrence visits at most $h + 1$ nodes in each of its k trees. The harmless $+1$ terms include leaves and empty roots. This bound excludes preprocessing and the cost of computing the tree assignments. It makes no balance assumption; replacing h by $O(\log N)$ requires an additional argument about tree shape.

Local update accounting. Consider an interval during which a node's candidate specification is unchanged. Let U_u be the number of signed sample updates delivered to it and E_u the number of full candidate reevaluations. A buffered update costs $O(1)$ when received and $O(p)$ when applied to

the histograms. Each full reevaluation costs $O(ps)$. Once pending histogram work is included, the total local cost is

$$O(U_u p + E_u ps), \qquad \text{or} \qquad O\left(p + ps\frac{E_u}{U_u}\right) \tag{11}$$

per delivered update when $U_u > 0$. Updates produced by migration count toward U_u just like external insertions and deletions. Initial construction, histogram regeneration, membership scans used to find movers, and new subtree construction are additional terms, not hidden in this expression.

For $\varepsilon = 0$, reevaluating after each singleton delivery costs $O(p + ps)$ per visited node. With no structural changes or specification refresh, an external request affecting k root-to-leaf paths therefore costs $O(k(h+1)(p+ps))$. Repairs may visit more nodes than these original paths.

Cost of a repair. A scan of a node's n current samples can identify the membership changes induced by a new split in $O(n)$ time. If m samples move between the two children, they cause m deletions and m insertions there. Updating reusable child histograms therefore costs $O(mp)$, with $O(ps)$ for their full candidate scans. The work at this level has the form

$$O(n + mp + ps). \tag{12}$$

Descendant repairs and any incompatible candidate state must be added. This is a local reuse bound, not a worst-case bound for the entire repaired subtree. Section D.1 explains when it improves on rebuilding and why $m \le n/2$ alone does not imply an asymptotic speedup.

The score-change bound used by deferred reevaluation. For completeness, we prove the Gini bound from the main text before discussing its implications for update cost.

Lemma B.5 (Gini score changes under modification)**.** *Let nonempty occurrence sets D and D' have sizes n and n'. Suppose D' is obtained through n_+ insertions and n_- deletions. For any fixed split* $\mathbf{s}$*,*

$$|\mathcal{I}(D, \mathbf{s}) - \mathcal{I}(D', \mathbf{s})| \le \frac{2\max\{n_+, n_-\}}{\max\{n, n'\}}.$$

Proof. Let P and P' be the uniform probability measures on the occurrence identifiers in D and D'. If $a = |D' \setminus D|$ and $b = |D \setminus D'|$, their total variation distance is

$$\mathrm{TV}(P, P') = 1 - \frac{|D \cap D'|}{\max\{n, n'\}} = \frac{\max\{a, b\}}{\max\{n, n'\}} \le \frac{\max\{n_+, n_-\}}{\max\{n, n'\}}.$$

For each side of $\mathbf{s}$, choose a probability vector over classes, denoted q_{L} or q_{R}. With e_y the unit vector for label y and $b_{\mathbf{s}}(\mathbf{x})$ the side selected by the split, weighted Gini impurity has the representation

$$\mathcal{I}(D, \mathbf{s}) = \min_{q_{\mathrm{L}}, q_{\mathrm{R}}} \mathbb{E}_{(\mathbf{x},y)\sim P} \left\| e_y - q_{b_{\mathbf{s}}(\mathbf{x})} \right\|_2^2.$$

Indeed, on each side the minimum is attained by its empirical class distribution and equals its Gini impurity. The squared loss lies in $[0, 2]$, so, for any fixed pair of probability vectors, changing P to P' changes the expected loss by at most $2\,\mathrm{TV}(P, P')$. Taking minima preserves this uniform bound. Substituting the bound on total variation proves the claim. □

For a fixed candidate set, Section D.2 uses this lemma to derive a certified update interval after installing an optimal split. Its scale is $\varepsilon n^{1-\varphi}$ when this quantity is between a constant and order n. If *every* reevaluation has an interval of at least B delivered updates, Eq. equation 11 gives $O(p+ps/B)$ amortized local cost, up to initialization and final-boundary terms.

The distinction is important: installing an optimal split resets its gap to zero, whereas reevaluating and retaining a suboptimal split does not. The stated retention rule alone therefore does not prove that every reevaluation is separated by $\Omega(\varepsilon n^{1-\varphi})$ updates. Without an additional slack-restoring condition, Eq. equation 11 is the general accounting bound; $O(p + ps/(\varepsilon n^{1-\varphi}))$ is conditional on such spacing.

Prediction. After required repairs have been completed, a forest query costs $O(T(h+1))$ for a fixed number of classes. If a query triggers deferred work, that work is added to its cost. Deferring an update changes when it is paid for, not whether it is paid for.

B.4 Memory Complexity

Each candidate attribute uses $s+1$ total bin counts and $s+1$ counts for each of c classes. Histograms therefore require $O(p(s+1)(c+1))$ words per maintained node; storing prefix sums separately changes only the constant factor. Installed-split statistics, candidate metadata, and child links add at most this order, plus a constant amount of node metadata.

Let R be the number of stored membership references, B the number of buffered signed-update entries, and W the maximum simultaneous repair workspace. Let N_{stored} count the distinct records still held in memory, including deleted records retained for pending work. If dense d-dimensional records are stored once and shared by reference, total memory is bounded by

$$O(N_{\text{stored}}d + T + Vp(s+1)(c+1) + R + B + W)\,. \tag{13}$$

For fixed c, the histogram term is $O(Vp(s+1))$. The record term reduces to $O(Nd)$ only when inactive records are no longer retained.

The value of R depends on the representation. Keeping memberships explicitly at every node gives $R = O(kN(h+1))$. Keeping occurrence references only at leaves gives $R = O(kN)$, but internal memberships then need to be recovered when a repair scans them. These are alternative storage bounds, not a claim that the supplied implementation uses either layout exclusively.

Likewise, the draft does not specify a global bound on pending-update storage or simultaneous migration buffers, so B and W should remain explicit. Tree subsampling reduces the total root membership to kN rather than TN, but does not by itself guarantee an identical proportional reduction in node count or process memory. Peak process RSS additionally includes input loading, runtime overhead, and temporary allocations.

C EXPERIMENTS

This section collects the setup and supporting results for Figures 4–7. We keep three questions separate: how well the model predicts, how closely it agrees with retraining, and how much computation it requires. Similar accuracy is not an exactness test, and fast initial training is not necessarily fast online maintenance.

C.1 BASELINES

We group the baselines by the operations evaluated. For mixed insertion and deletion, we compare Sparse Forest with DynFrs (Wang et al., 2025) and DyGB (Lin et al., 2025). DynFrs is the randomized-forest baseline and uses $q = 0.2$; DyGB maintains a gradient-boosted tree model. For deletion-only workloads, we compare with DaRE (Brophy and Lowd, 2021) and HedgeCut (Schelter et al., 2021). For insertion-only learning, we compare the single-tree Sparse Tree configuration, $T = k = 1$, with VFDT and EFDT (Manapragada et al., 2018).

We use each baseline's supported update interface rather than presenting unsupported operations as native updates. Single trees and forests are compared separately. The selected configurations are practical settings, not an equal-capacity comparison, so their runtime differences should not be attributed to one sparsity mechanism alone.

C.2 DATASETS

The main experiments use nine binary-classification datasets spanning census records, marketing, fraud detection, healthcare, e-commerce, and particle physics. The additional task experiments in Appendix C.5 use three multiclass-classification and two regression datasets, described below.

Binary classification.

Adult (Becker & Kohavi, 1996) is derived from the 1994 U.S. Census database. The task is to predict whether an individual's annual income exceeds $50,000 from demographic and employment attributes, such as age, education, occupation, and working hours.

Bank (Moro et al., 2014a;b) contains records of telephone marketing campaigns conducted by a Portuguese bank. The target indicates whether a client subscribes to a term deposit, using client characteristics and information about the campaign and previous contacts.

CreditCard (Dal Pozzolo et al., 2015) contains European cardholders' transactions over two days in September 2013. Its 30 predictors comprise 28 anonymized principal components, transaction time, and amount; the target identifies fraudulent transactions. We use OpenML dataset 1597, version 1,[1] with a fixed stratified 80/20 split of 284,807 transactions, including 492 fraud cases. Majority-class accuracy is about 99.83%, so accuracy must be read together with AUC.

Diabetes (classification) (Clore et al., 2014; Strack et al., 2014) contains hospital encounters involving patients with diabetes across 130 U.S. hospitals and integrated delivery networks during 1999–2008. It supports hospital-readmission prediction from patient demographics, diagnoses, laboratory measurements, and treatment information. This clinical-record dataset is distinct from the Diabetes regression dataset below.

Heart (Ulianova, 2018) is the Cardiovascular Disease dataset, containing 70,000 patient records. Its binary target indicates the presence of cardiovascular disease, with predictors describing demographics, physical measurements, blood pressure, laboratory indicators, and lifestyle factors.

Higgs (Baldi et al., 2014) contains 11 million simulated particle-collision events. The task is to distinguish Higgs-boson signal events from background using 28 numerical features: 21 low-level kinematic measurements and seven derived high-level quantities. Our split contains 8.8 million training and 2.2 million TEST examples.

[1] https://www.openml.org/d/1597

NoShow (Hoppen & Aquarela Analytics, 2016) contains 110,527 medical appointments in Brazil. The task is to predict appointment attendance from patient and appointment information, including age, health conditions, scheduling dates, and whether an SMS reminder was received.

Purchase (Sakar & Kastro, 2018) is the Online Shoppers Purchasing Intention dataset, containing 12,330 website sessions. The binary target records whether a session ends in a purchase; predictors describe browsing activity, page durations, traffic source, and visitor characteristics.

Vaccine (DrivenData, 2019) comes from DrivenData's Flu Shot Learning competition and contains 26,707 labeled responses to the National 2009 H1N1 Flu Survey. We predict receipt of the seasonal flu vaccine from respondents' demographic information, health behaviors, and vaccination-related opinions.

Multiclass classification and regression. Iris, Wine, Digits, and regression Diabetes use the datasets distributed with scikit-learn (Pedregosa et al., 2011); Auto MPG is obtained from UCI.

Iris (Fisher, 1936) contains 150 flowers from three species. Four measurements—sepal length and width and petal length and width—are used to predict the species.

Wine (Aeberhard & Forina, 1992) contains 178 wines from three cultivars grown in the same Italian region. The task is to identify the cultivar from 13 chemical measurements, including alcohol, acidity, and phenolic composition.

Digits (Alpaydin & Kaynak, 1998) contains 1,797 handwritten digit images in the scikit-learn subset of UCI's Optical Recognition of Handwritten Digits dataset. Each example has 64 features representing an 8×8 image, and the target is one of the ten digits, 0–9.

Diabetes (regression) (Efron et al., 2004) contains 442 patients with ten baseline predictors: age, sex, body mass index, average blood pressure, and six blood-serum measurements. The continuous target measures disease progression one year after baseline.

Auto MPG (Quinlan, 1993) predicts city-cycle fuel economy in miles per gallon from vehicle characteristics. We retain 392 complete records after removing six rows with missing horsepower, omit car names, and use seven predictors: cylinders, displacement, horsepower, weight, acceleration, model year, and origin.

Data partitions and model selection. Preprocessing is fitted on training data, and TEST is not used for model selection. The additional multiclass and regression experiments use the separate fixed 70/30 development/TEST protocol in Appendix C.5.

The seven smaller binary-classification datasets (Adult, Bank, Diabetes, Heart, NoShow, Purchase, and Vaccine) use five-fold training-only selection. The recorded model-specific grids contain 90 configurations, or 30 distinct configurations for single trees after fixing $T = k = 1$. Selection maximizes validation AUC on Bank, NoShow, and Purchase, and validation accuracy on the remaining smaller datasets. The other metric and a fixed configuration order break ties; all five folds must be valid. HedgeCut uses fixed defaults on Adult and Vaccine because its selection runs did not produce a configuration with five valid folds.

The three main online comparisons use all nine datasets. Other experiments use the fixed subsets in Table 1; they should not be described as nine-dataset aggregates.

C.3 Implementations

Model configurations. The recorded Sparse Forest implementation is v3.3.2. CreditCard and Higgs use fixed settings: 100 trees for the forest models; depth 25, 40 thresholds, and $k = 20$ for SF; depth 20, 30 thresholds, and $k = 20$ for DynFrs; and depth 20, 25 thresholds, and no random top levels for DaRE. HedgeCut uses minimum leaf size 10 and 10 split attempts. DyGB uses 100 boosting iterations, 20 leaves, and shrinkage 0.1. On the smaller datasets, SF's exact and split-retention configurations may be selected separately; the tolerance ablation instead fixes the remaining configuration.

Comparison	Dataset subset
Online comparisons	All nine datasets
Initial predictive performance	Bank, Diabetes, Heart, NoShow, Purchase
Training: single trees	All datasets except Higgs
Training: forests	Bank, Heart, NoShow, Purchase
Sequential insertion: forests	Adult, CreditCard, Purchase
Sequential insertion: single trees	Adult, Bank, CreditCard, Diabetes, Vaccine
Split-retention ablation	Adult, Bank, Diabetes, Heart, Vaccine

Table 1: Fixed dataset subsets used for aggregation. Retraining comparisons report dataset-level results, with full-data and reduced-scale tests distinguished in the text.

Exact and positive-tolerance settings. The recorded tables call the native exact configurations SF0 and ST0, and the native $\gamma = 0.05$ configurations SF.05 and ST.05. These latter labels are implementation parameters, not the normalized ε in the suboptimality definition. The native score is $n\mathcal{I}/2$, so a raw margin of $0.05\sqrt{n}$ corresponds to normalized tolerance $0.1/\sqrt{n}$.

Figure 6 removes that floor and tests the pure rule $\varepsilon n^{-\varphi}$ for $\varphi \in \{0, 1/2, 1\}$ and $\varepsilon \in \{0.025, 0.05, 0.1, 0.2, 0.4\}$, together with a zero-tolerance control. That control retains the positive-γ candidate-generation rule, including adaptive redraw. It is not native SF0, which fixes path randomness and canonical ties and disables adaptive redraw.

Request streams. Mixed streams contain 50,000 insertions, 50,000 deletions, and 100,000 queries. Insertion-only and deletion-only streams each contain 100,000 updates and 100,000 queries. For mixed and insertion workloads, TRAIN is divided into disjoint 70% initial, 20% arrival, and 10% query populations. Repeated arrivals receive new occurrence identifiers, and deletions refer only to live occurrences. Deletion workloads concatenate episodes; each starts from the 90% non-query population and deletes at most half of it.

Predictive performance. Initial performance is measured after training on TRAIN and evaluating on TEST. Online performance uses held-out stream queries that never enter training; it is not prequential accuracy on arriving training samples. Terminal TEST evaluation is a separate endpoint. Figure 4 combines accuracy and AUC as $(\mathrm{ACC} + \mathrm{AUC})/2$, while the underlying results keep the two metrics separate.

Efficiency measurements. Runs use one pinned physical CPU core and one native thread. Request-processing time includes the updates and queries served during the stream, but excludes initial training, episode resets, final drain, and TEST evaluation. Sequential and batch-update measurements include final repair. Consequently, the main stream-time endpoint measures work paid during the stream, not the total cost of fully materializing the final model.

Figure 4 uses observed same-dataset, same-workload, same-seed time ratios without requiring matched timing blocks. The retention, sequential, and batch comparisons use paired records with a common host, core, seed, and timing block, cycling method order. Their timing admission requires a CPU/wall ratio between 0.90 and 1.05, mean SMT-sibling activity at most 2%, and peak activity at most 10%. Training summaries use admitted build measurements. These are implementation comparisons; they do not isolate algorithmic complexity from runtime and representation differences.

Aggregation. Within a fixed dataset subset, predictive metrics are averaged arithmetically and time ratios geometrically for each seed. Build time and RSS also use geometric means within a seed. Reported points then average the seed-level scores. Where repeated complete seeds are available, 95% intervals use 10,000 bootstrap resamples of seeds, not datasets. A seed enters an aggregate only when every dataset in that subset is observed for that metric. A single seed gives a point without a seed-variation interval. Projected entries receive no synthetic confidence interval.

Baseline-specific qualifications. DynFrs uses current candidate scores rather than historical score minima. Its original mixed-stream runs produced finite online predictions but invalid terminal states; the plotted Higgs checkpoint recorded all requested operations before a later timeout. The corre-

sponding Figure 4 coordinate is therefore an observed execution, not evidence of a valid completed exact-maintenance run. Figure 5 instead uses the corrected deferred-drain branch, which clears pending state and rechecks split eligibility through the normal build path.

DyGB's native state reload and update work is included in its timing. Its reduced-scale retraining diagnostic uses native batches of sixteen label replacements, not singleton interleaving. This schedule was fixed across all nine datasets after an earlier schedule was found to rebuild every root at the final batch on Bank and Purchase, giving zero terminal KL despite earlier divergence. The batch diagnostic must therefore be distinguished from both that earlier control and the stream-runtime experiment.

HedgeCut uses training-fitted quantization and tree-vote probabilities. A validated prediction-only builder is used for some fresh references, and one reduced-scale Vaccine maintenance trajectory uses a validated deferred executor. These substitutions are used only to recover predictive trajectories; their execution times and memory are not reported as native baseline costs. The deferred executor retains records needed for reconstruction and is not a separate erasure guarantee.

C.4 More Results

Predictive performance and completed deletion runs. The stream comparisons show similar aggregate predictive performance for the selected SF configurations, but that does not mean SF is best on every dataset or metric. For example, VFDT has higher initial TEST AUC than ST.05 on CreditCard: 93.80% versus 88.25%. Table 2 retains the completed deletion comparisons and makes missing full-stream measurements explicit.

Figure 8 gives the initial TEST comparison on the common five-dataset subset, independently of update timing.

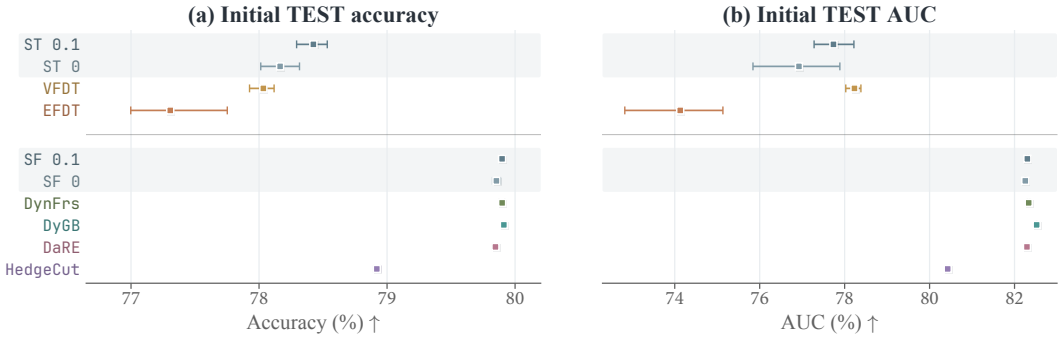


Figure 8: Initial TEST accuracy and AUC on Bank, Diabetes, Heart, NoShow, and Purchase. Points are means of five seed-level dataset averages; bars are 95% seed-bootstrap intervals. Single-tree and forest capacities are shown separately. The plot labels ST 0.1 and SF 0.1 denote the native ST.05 and SF.05 configurations described in Section C.3, not the pure tolerance rule of Figure 6.

Estimated entries in Figure 4. Not every baseline completed every full workload. DyGB has no completed full mixed-query stream across the nine datasets; DaRE and HedgeCut have completed deletion streams on seven and five datasets, respectively. Missing runtime entries use measured native-operation prefixes, and some also require population extrapolation. Prefix projection assumes that observed mean operation costs remain representative over the full request count. Population projection fits nonnegative log–log slopes to operation costs measured on smaller populations. These assumptions are exploratory. In particular, HedgeCut's CreditCard projection rests on only 32- and 64-record pilots and is too distant and unstable to support a precise speedup claim.

Missing stream-performance entries use the baseline's initial TEST performance plus the mean initial-to-stream change of SF0 and SF.05 on that dataset, clipped to the valid range. This transfers a performance change across learners and is not a measured online result. The resulting nine-dataset estimates are 83.85% for DyGB, 83.60% for DaRE, and 82.39% for HedgeCut. DyGB's transfer has no completed-stream validation. The Higgs initial-quality anchors for DaRE and HedgeCut are

	SF.05	DaRE			HedgeCut		
Dataset	Perf.	Seeds	Time ratio	Perf.	Seeds	Time ratio	Perf.
Adult	89.26	5	65.18	89.18	0	–	–
Bank	93.06	3	222.11	92.79	5	2.78	91.52
CreditCard	98.07	0	–	–	0	–	–
Diabetes	67.61	2	106.43	67.45	2	19.68	65.24
Heart	76.68	5	88.50	76.55	5	2.35	76.17
Higgs	78.63	0	–	–	0	–	–
NoShow	76.46	2	125.48	75.94	5	2.33	74.90
Purchase	91.14	3	162.05	91.47	5	2.51	90.49
Vaccine	83.53	1	781.95	82.59	0	–	–

Table 2: Completed deletion streams. Performance is $(\mathrm{ACC} + \mathrm{AUC})/2$ in percent. SF.05 averages five seeds; the baseline seed counts are shown. Time ratios compare each baseline with SF.05 on the same dataset and seed, then average the available complete seeds. Dashes are missing measurements, not zeros. The entries reproduce the supplied manuscript's completed-stream table.

themselves learning-curve estimates from smaller training populations. These entries must remain visibly marked as estimates and must not establish a predictive ranking.

Agreement with retraining. The retraining diagnostic fixes a half-TRAIN label-corruption mask and runs twenty trials. Mixed updates replace the selected records by deleting them and reinserting them with flipped binary labels. Insertion starts with the clean half and adds the flipped half. Deletion starts with both halves and removes the corrupted half. Maintained and freshly trained models are compared at five checkpoints. Each reference uses that learner's own builder, preprocessing, hyperparameters, and specified randomness or replay order.

For reference and maintained predictive distributions $p_r(\mathbf{x})$ and $q_r(\mathbf{x})$ in trial r, the recorded summaries are

$$K = \frac{1}{20}\sum_{r=1}^{20}\frac{1}{|\mathrm{TEST}|}\sum_{\mathbf{x}\in\mathrm{TEST}} D_{\mathrm{KL}}\big(p_r(\mathbf{x})\|q_r(\mathbf{x})\big), \tag{14}$$

$$K_{\mathrm{avg}} = \frac{1}{|\mathrm{TEST}|}\sum_{\mathbf{x}\in\mathrm{TEST}} D_{\mathrm{KL}}\left(\frac{1}{20}\sum_r p_r(\mathbf{x})\,\middle\|\,\frac{1}{20}\sum_r q_r(\mathbf{x})\right). \tag{15}$$

For binary predictions, these are KL divergences between Bernoulli distributions. Figure 5 uses K_{avg}. Averaging predictions first can hide substantial trial-level differences: for ST.05 on Adult, $K_{\mathrm{avg}} = 0.00705$ whereas $K = 0.84009$ nats. Probabilities are clipped at 10^{-12} for finite summaries; unmatched supports can make the unclipped KL infinite. Neither a small predictive KL nor zero KL on a finite TEST set proves equality of model distributions.

Native SF0 matches fresh prediction bits and canonical model state on eight full-data datasets under mixed updates and deletion. Those two full-data Higgs cells are not covered: the mixed run encountered allocation failure under the recorded limit, and neither setting has a valid comparison there. ST0 matches on all nine full-data insertion datasets.

To complete the all-dataset diagnostics in Figure 5, mixed updates use 256 training records and deletion uses 2,048, with a common fixed TEST sample of at most 2,048 records. Training samples are class-proportional. These reduced-scale tests complete twenty trials per method and dataset; SF0 matches prediction bits and canonical state at all 1,800 paired checkpoints across its two panels. This extends coverage across datasets, not across full training populations.

VFDT and EFDT exactly reproduce their predictions under same-order replay. Figure 5 additionally uses independently shuffled replay of the same active records. The nonzero divergence there measures insertion-order sensitivity, including the effect of changing the clean-then-corrupted presentation order; it is not an unlearning test. The corrected DynFrs, batched DyGB, and HedgeCut reference procedures described above apply to these reduced-scale diagnostics rather than retroactively replacing the original stream timings.

Split retention. Figure 6 holds the remaining selected configuration fixed and varies only the pure tolerance rule described in Section C.3. On its five-dataset subset, $(\varepsilon, \varphi) = (0.1, 0)$ uses 0.534 times the native SF.05 processing time, with AUC 84.71% versus 84.63%. Its mean K_{avg} increases from 8.56×10^{-4} to 1.54×10^{-3} nats. Thus the observed benefit is faster maintenance with little predictive change but less agreement with retraining.

Training time and memory. Table 3 gives selected absolute resource measurements behind Figure 7. Peak RSS includes process memory, not only the model. On full Higgs, ST.05 builds faster than VFDT, 43.27 versus 62.41 seconds, but uses more peak RSS, 5681.5 versus 3606.8 MiB. Thus lower memory use is not universal. Exact SF also builds faster than SF.05 on the forest subset, despite taking longer to process the online streams. Initial training time alone is therefore an incomplete efficiency measure.

Dataset subset	Model	Build time (s)	Peak RSS (MiB)
Eight-dataset single-tree subset	ST.05	0.092	62.1
	VFDT	1.846	314.2
	EFDT	3.429	416.9
Four-dataset forest subset	SF.05	0.754	146.6
	DynFrs	2.111	386.5
	DyGB	4.669	3100.3

Table 3: Selected training-resource measurements. Dataset subsets are listed in Table 1. These are aggregate absolute measurements, not isolated model allocations or paired host ratios. RSS is reported in MiB here.

Other update workloads. Sequential insertion removes interleaved queries and includes all final repair work. Over Adult, CreditCard, and Purchase, SF.05, SF0, and DynFrs have relative times 1.00, 1.86, and 5.86. Over the five-dataset single-tree subset, ST.05, ST0, VFDT, and EFDT have ratios 1.00, 3.46, 4.62, and 9.78. There is no corresponding five-seed admitted deletion aggregate.

Figure 9 shows these insertion timings and their uncertainty.

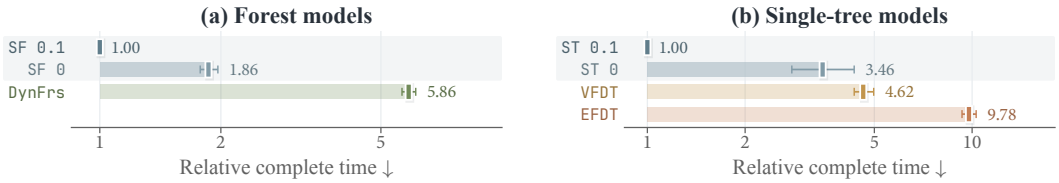


Figure 9: Complete sequential insertion time, including final repair and without interleaved queries. Forest results use Adult, CreditCard, and Purchase; single-tree results use Adult, Bank, CreditCard, Diabetes, and Vaccine (Table 1). Ratios use the faster admitted native SF setting (ST for single trees) within each paired seed and timing block. Points and 95% seed-bootstrap intervals summarize five seed-level scores. There is no corresponding admitted five-seed deletion aggregate.

For native batch insertion of 1% of Higgs TRAIN, SF.05 and SF0 take 16.72 and 111.02 seconds including final repair. The mean paired ratio is 7.04, which differs from the ratio of those mean times. This is one batch-size comparison, not a measured scaling law. A separate shorter mixed Higgs stream, containing 5,000 insertions, 5,000 deletions, and 10,000 queries, gives relative request-processing times 1.00, 27.64, and 106.78 for SF.05, SF0, and DynFrs. It should not be merged with the full-length workload.

Figure 10 reports the native batch comparison; Figure 11 keeps the shorter mixed stream separate.

Batch scaling on Higgs. Figure 12 varies the batch size from one to 10^5 records on a fixed Higgs subset. Each size starts from a newly built model: insertion grows the active population from $200{,}000 - B$ to 200,000, while deletion reduces it from 200,000 to $200{,}000 - B$. Requests preserve the original labels, and each fresh reference uses the same final active data. This subset study is

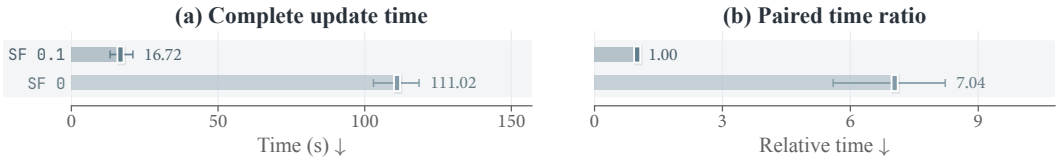


Figure 10: Higgs native batch insertion at 1% of TRAIN. Complete update time includes final repair. Panel (a) shows mean seconds; panel (b) averages paired ratios relative to the faster native SF setting within each seed. These are different summaries: the mean paired ratio is not the ratio of mean times. Bars are 95% seed-bootstrap intervals over five seeds. This single batch size does not establish a scaling curve.

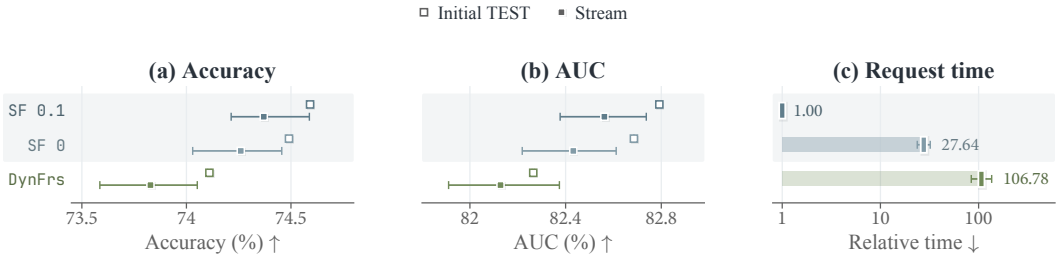


Figure 11: Shorter mixed Higgs stream with 5,000 insertions, 5,000 deletions, and 10,000 queries. Open and filled quality markers denote initial TEST and stream quality, respectively. Request-processing time is relative to native SF.05 and excludes initial training, final drain, and terminal TEST evaluation. Points are five-seed means; bars are 95% seed-bootstrap intervals. These measurements are separate from the full-length workload in Figure 4.

separate from both the full-TRAIN batch measurement above and the label-replacement diagnostic in Figure 5.

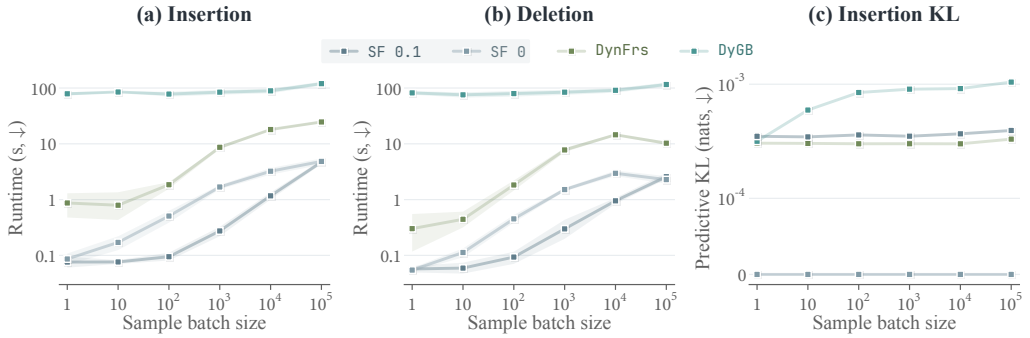


Figure 12: Batch scaling on a fixed stratified Higgs subset of 200,000 TRAIN and 20,000 TEST records. Panels (a,b) show mean completed batch runtime over five seeds, including updates and final drain; light bands are pointwise 95% seed-bootstrap intervals. Panel (c) shows insertion-only K_{avg}, computed by averaging fresh and maintained probabilities separately over twenty seeds before evaluating TEST-mean Bernoulli KL (fresh ‖ maintained). No KL uncertainty band is shown. DynFrs runtime at 10^5 deletions is retained despite invalid final predictions.

C.5 Multiclass Classification and Regression

We evaluate Iris, Wine, and Digits classification (3, 3, and 10 classes; 150, 178, and 1,797 examples), and Diabetes and Auto MPG regression (442 and 392 examples). Auto MPG drops six incomplete rows and car names; regression Diabetes differs from the binary dataset above. Scikit-learn Random Forest is the only baseline. All arms use 100 trees, depth 16, and one thread. Each learner receives twelve configurations and three-fold initial-TRAIN selection by accuracy or RMSE; both Sparse Forest settings share the winner. A fixed 70/30 development/TEST split keeps TEST untouched. A disjoint stream reserves 21–32 development records: each is queried, inserted, and followed by deletion of a distinct initial-TRAIN record.

This task variant uses Gini or squared-error splits, 32 training-fitted feature bins, and eager local recount and child-population repair; regression targets remain real-valued. It omits the binary engine's lazy certificates and adaptive redraw. Here, an incumbent stays within an impurity-gain deficit of γ times parent impurity, a separate rule from the native binary margin above.

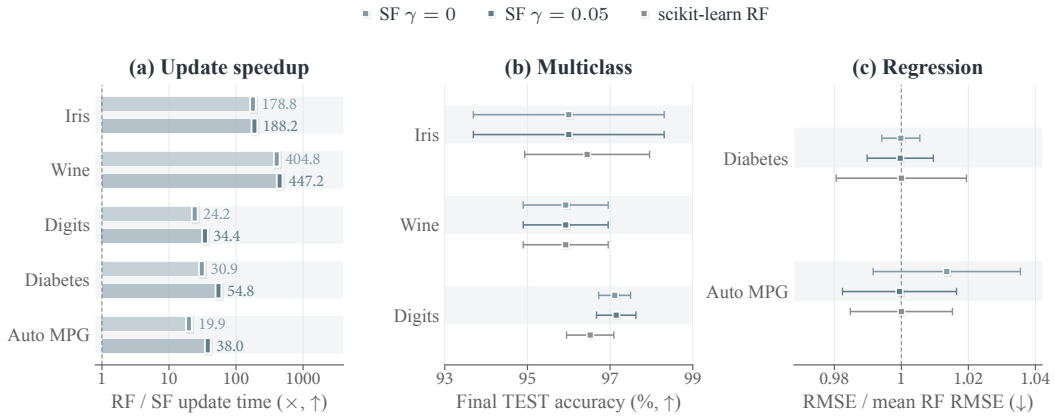


Figure 13: Small-data task generalization over five paired model/deletion-order seeds. Panel (a) reports geometric mean RF/SF insertion-plus-deletion time ratios; RF refits after every operation. Panels (b,c) report final TEST performance. Bars are 95% t intervals, using log ratios for speedup. In (c), each RMSE and its interval are divided by the fixed observed mean RF RMSE. Intervals describe seed variation on one fixed split, not uncertainty over resampled datasets.

Figure 13 shows update speedups of 19.9–404.8× at $\gamma = 0$ and 34.4–447.2× at $\gamma = 0.05$, under immediate per-operation consistency. Predictive results are mixed: Digits accuracy rises from 96.52% to 97.11/97.15%, Wine ties, and Iris loses 0.44 percentage points. Regression RMSE is close, except for a 1.36% increase on Auto MPG at $\gamma = 0$. Accuracy-focused tuning also leaves worse Iris log loss (0.993/0.840 versus RF's 0.111) and worse Wine log loss. Faster updates do not imply faster batch inference: final Digits TEST prediction is about 1.80× slower than RF.

After every insertion and deletion, we compare with a fresh task forest using the same active multiset, frozen encoding, configuration, and seed, with reversed input order. At $\gamma = 0$, all 1,410 checkpoints match both full logical state and TEST prediction bits. At $\gamma = 0.05$, only 1/1,410 state checks and 41/1,410 prediction-bit checks match. This is same-configured task-learner equality at zero tolerance; positive tolerance is not retraining-exact.

D Discussion

D.1 In-Place Repair Gain

The main advantage of in-place repair is simple: a changed parent split does not make every stored statistic useless. Samples that remain in a child need not be removed from and reinserted into its compatible histograms. Repair pays for changed membership; rebuilding pays for all membership.

For the same parent sample set before and after a split change, write the four intersections as in the main text. Keeping the existing child assignment moves

$$m_{\text{keep}} = |A_{01}| + |A_{10}|,$$

whereas exchanging the assignment moves

$$m_{\text{swap}} = |A_{00}| + |A_{11}|.$$

These two quantities add to n. When both assignments are permitted, choosing the smaller therefore gives

$$m = \min\{m_{\text{keep}}, m_{\text{swap}}\} \le n/2. \tag{16}$$

For a fixed parent population, each moved sample produces one deletion and one insertion, so the children receive $2m$ signed membership changes. Parent insertions and deletions are additional changes; Algorithm 3 handles them by comparing the previous child sets directly with the final targets.

What the bound buys. Rebuilding the two children processes all n samples across their candidate attributes, whereas reuse processes the $2m$ changes. Including the scan that identifies movers gives the local comparison

$$\text{rebuild: } O(np + ps), \qquad \text{repair: } O(n + mp + ps).$$

The factor two in $2m$ is absorbed by big-O, but matters when interpreting savings. At $m = n/2$, the child histogram-update count can be as large as rebuilding, before adding the membership scan. At $m \ll n$, substantial statistics work can be avoided, especially when descendants also remain reusable.

The main text reports average mover fractions of roughly $0.37n$ without exchanging children and $0.26n$ with exchange. These are membership-change measurements, not wall-clock speedup factors. Their benefit depends on where changes propagate, how much time is spent scanning samples and candidates, and whether node construction state remains compatible.

What it does not buy. The bound does not guarantee a constant-factor reduction in total repair time, let alone an asymptotic improvement for every update. A repair can cause further split changes below the current node; shallow nodes may spend more time finding movers than rebuilding their small children; and an exchanged child may require candidate regeneration to preserve total exactness. If only one child assignment is compatible with the chosen implementation, the $n/2$ guarantee does not apply to that restricted choice.

The practical question is therefore not only how many samples move at the first changed split, but how much work remains reusable throughout the affected subtree. A mechanism-level comparison should count membership changes, bin updates, full candidate scans, and reconstructed nodes, as well as complete repair time.

Measured statistics workload. Figure 14 measures the sample contributions needed to maintain split statistics throughout a repair cascade. One contribution applies one sample to one node's statistics in one update or construction pass, without multiplying by the number of candidate attributes or thresholds. Thus a transfer between siblings contributes twice, once as a deletion and once as an insertion. At relative layer ℓ, we count

$$N_{\text{repair},\ell} = D_\ell + B_\ell + G_\ell, \qquad R_\ell = \frac{\sum_e N_{\text{repair},e,\ell}}{\sum_e N_{\text{rebuild},e,\ell}}, \tag{17}$$

where D_ℓ counts deltas actually applied to existing statistics, B_ℓ counts statistics construction in rebuilt subtrees, and G_ℓ counts construction when leaves grow. The sums pool repair cascades e.

Canceled or discarded deltas contribute zero; a sample processed in successive passes contributes in each pass. Candidate-redraw passes, membership scans, feature masks, split scoring, and terminal-leaf label counts are excluded from this workload measure.

The reference $N_{\mathrm{rebuild},e,\ell}$ counts one contribution per live sample at every split-eligible node in layer ℓ of the *same final subtree*, including untouched branches. Nodes stopped by purity, minimum support, or maximum depth contribute zero. This holds the final population and topology fixed; it does not run a separate rebuild-only learner that could select different splits. Layer 0 is the node whose installed routing predicate changes. A cascade starts when this change is handled in place and includes subsequent rebuilding and growth below it; standalone root rebuilds are outside this conditional sample.

Diagnostic setup. We profile v3.3.2 on Adult, Bank, NoShow, Diabetes, Heart, Purchase, and Vaccine, using three seeds per native setting, $\gamma = 0$ and 0.05 (Section C.3). Each run starts with 8,192 training samples and processes 512 disjoint insertions and 512 deletions of live samples. We fix 100 trees, $k = 10$, maximum depth 20, minimum split support 10, 11 candidate attributes, 30 thresholds, and a fixed 256-bin encoding. Draining after each edit isolates complete cascades, rather than allowing several edits to coalesce. The resulting 84,401 and 16,215 cascades pass all 43,008 post-edit state comparisons with the uninstrumented model, together with the prediction checks.

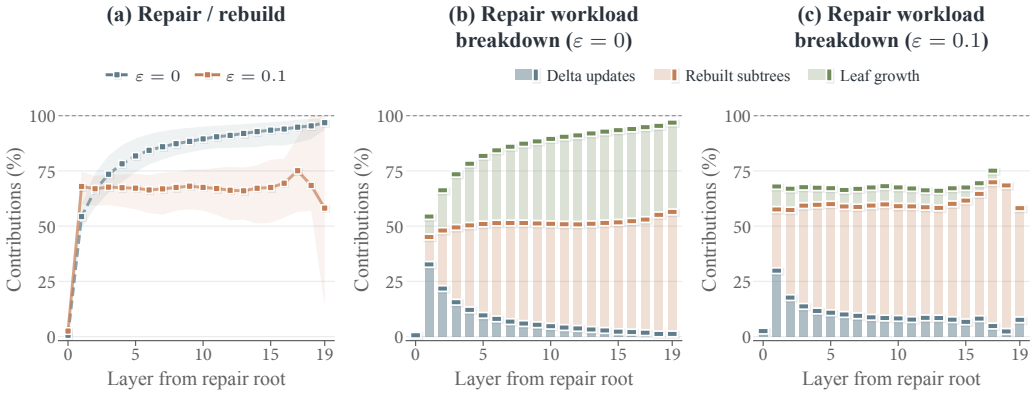


Figure 14: Split-statistics workload of in-place repair, with layers numbered from 0 at the changed split. Panel (a) reports the ratio of pooled repair contributions to full statistics reconstruction on the same final subtree; lower is better and 100% denotes the reference workload. Light bands are pointwise descriptive 95% intervals from 10,000 hierarchical dataset/seed bootstrap resamples. Panels (b,c) decompose the numerator into delta updates, rebuilt subtrees, and leaf growth, using the same layer-specific denominator. Plot labels use $\varepsilon = 2\gamma$ for the native settings. Candidate redraws are excluded. These are sample-contribution counts; scanning, scoring, and elapsed time are outside the measure.

What the profile shows. Summed across all layers, the repair-to-rebuild ratio is 67.19% at $\gamma = 0$ and 53.26% at $\gamma = 0.05$, corresponding to 32.81% and 46.74% fewer contributions in the measured workload. The descriptive 95% intervals for these ratios are 61.27–70.36% and 47.05–57.42%, respectively. The top-layer ratios are only 0.63% and 2.53% because the top node reuses its already-maintained statistics; its movers primarily generate updates in the children. At layer 1, the ratios rise to 54.34% and 67.95%.

Deeper repair is increasingly dominated by constructing statistics in rebuilt or newly grown subtrees. At $\gamma = 0$, the ratio reaches 96.82% at layer 19: rebuilding and growth contribute 95.58 percentage points, while delta updates contribute only 1.24. A rapidly falling count of *new* movers therefore does not imply equally rapid decay in statistics work. Inherited deltas can update a node whose split remains unchanged, and rebuilding continues processing samples after retained repair ends. The reconstruction denominator also shrinks as samples reach terminal leaves. The deepest $\gamma = 0.05$ point has only 184 reference contributions from six cascades, so its fluctuation warrants little weight. Overall, reuse saves substantial contributions in this diagnostic, with most of the relative gain near

the repair root; the count reduction does not establish an elapsed-time speedup or a uniform per-update bound.

D.2 Reevaluation Frequency

Reevaluation checks whether the installed split remains satisfactory; replacement changes the structure only if it does not. A failed upper-bound check triggers reevaluation, not automatically replacement.

A safe test between reevaluations. Let D_0 be the node population at the last reevaluation, with $n_0 = |D_0|$ and best score $\mathcal{I}_0^\star$. After n_+ insertions and n_- deletions, write D for the current population and $n = |D| > 0$. While the candidate set is unchanged, the Gini bound gives

$$\min_{\mathbf{s}\in C} \mathcal{I}(D, \mathbf{s}) \geq \mathcal{I}_0^\star - \delta, \qquad \delta = \frac{2\max\{n_+, n_-\}}{\max\{n_0, n\}}.$$

Consequently,

$$g = \mathcal{I}(D, \hat{\mathbf{s}}) - \min_{\mathbf{s}\in C} \mathcal{I}(D, \mathbf{s}) \leq \hat{g} := \mathcal{I}(D, \hat{\mathbf{s}}) - \mathcal{I}_0^\star + \delta. \tag{18}$$

If $\hat{g} \leq \varepsilon n^{-\varphi}$, retaining the split is safe; otherwise, the node reevaluates. With current installed-split counts and a fixed number of classes, the test costs $O(1)$ per delivered update. Changes to candidate or eligibility rules invalidate it. The proof is for Gini, not entropy.

Why the scale is $\varepsilon n^{1-\varphi}$. Write $K = n_+ + n_-$ and let g_0 be the gap just after the last reevaluation. While the installed split is unchanged,

$$\hat{g} \leq g_0 + 2\delta \leq g_0 + \frac{4K}{n_0}.$$

Also $n \leq n_0 + K$, so a sufficient condition for skipping reevaluation is

$$g_0 + \frac{4K}{n_0} \leq \varepsilon(n_0 + K)^{-\varphi}. \tag{19}$$

This condition accounts for the change in tolerance as the node grows.

Immediately after installing an optimal split, $g_0 = 0$. If

$$K \leq \left\lfloor \min\left\{ \frac{n_0}{2}, \frac{\varepsilon}{4(3/2)^\varphi} n_0^{1-\varphi} \right\} \right\rfloor, \tag{20}$$

then $n \leq 3n_0/2$, and Eq. equation 19 holds. For nontrivial blocks below the population-size cap, the interval scales as $\varepsilon n_0^{1-\varphi}$: linearly with n_0 at $\varphi = 0$, as $\sqrt{n_0}$ at $\varphi = 1/2$, and independently of n_0 at $\varphi = 1$. Small integer intervals may still require reevaluation at every update.

Why this is not a universal reevaluation interval. Reevaluation can retain a split with $0 < g_0 \leq \varepsilon n_0^{-\varphi}$. Near the tolerance, little slack remains in Eq. equation 19, and the next update may require another scan. Only replacement necessarily resets the gap to zero.

The zero-gap argument therefore protects an interval after replacement, not after every reevaluation. A uniform amortized rate based on $\varepsilon n^{1-\varphi}$ needs a slack-restoring rule or a separate amortization argument. Without one, the general bound remains $O(U_u p + E_u ps)$.

The $\sqrt{n}$ interval follows from the chosen tolerance and deterministic Gini bound, not from a claim that closer empirical scores are indistinguishable. Total exactness remains the separate zero-tolerance configuration under its fixed specification.

Empirical frequency on full Higgs. We compare the existing conservative trigger (*naïve trigger*) with Eq. 4, retaining the split-selection and margin rules. We build v3.3.2 on all 8.8 million Higgs training rows with $T = 100$, $k = 20$ ($q = 0.2$), $\varepsilon = 0.1$ ($\gamma = 0.05$), $\varphi = 1/2$, and depth 20. One seed processes 20,000 edits: blocks of up to 64 randomly selected training rows are deleted and reinserted, with one training query after each edit and a final full drain.

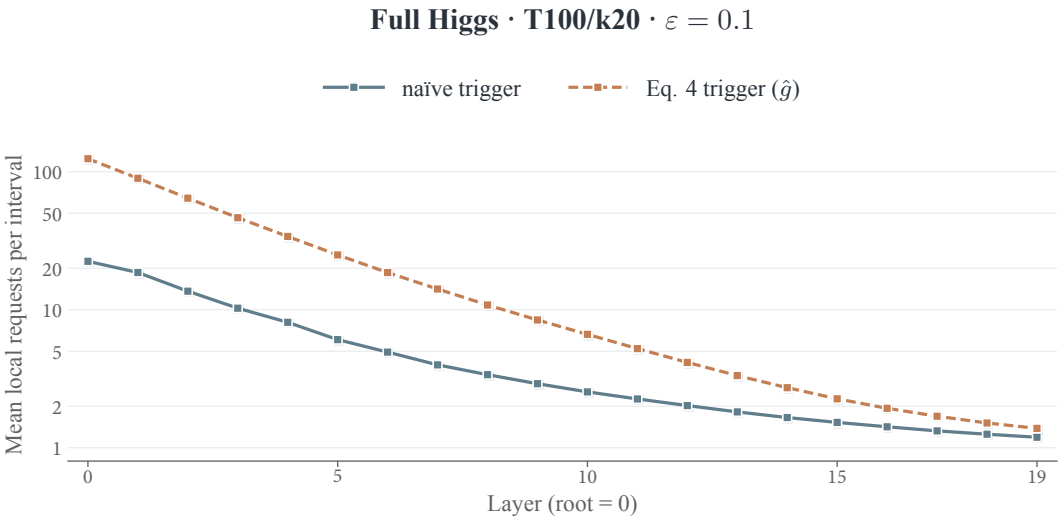


Figure 15: Decision-check spacing on full Higgs. At each layer, the curve is the event-weighted mean number K of local addition/deletion requests between consecutive maintenance decision checks in the same node and candidate-bank epoch. Incoming migration deltas count at the receiving node. Completed intervals include zero-request follow-up checks; first checks, bank resets, and unfinished intervals are excluded. Layer 0 is the root; the vertical axis is logarithmic. This one-seed profile measures decision-check frequency, not necessarily full candidate rescans.

Figure 15 shows that Eq. 4 increases the mean completed interval at the root from 22.4 to 124.2 local requests. The mean initial root population is 1.76 million, giving $\varepsilon\sqrt{n} = 132.7$. Across all layers, total decision calls decrease from 3,339,837 to 2,319,151 (30.6%). These totals include first checks and thus differ from the completed-interval population plotted in the figure. All 22,048 paired query predictions are bit-identical, and the final semantic-state hashes agree; this is an observation for this trace, not a general equivalence guarantee.

The root scale is consistent with balanced insertions and deletions: $2\max\{n_+, n_-\} \approx K$, so when the installed score changes little and remains near the previous optimum, $\hat{g} \approx K/n$ reaches the tolerance near $K = \varepsilon\sqrt{n}$. Deeper nodes contain fewer samples, reducing this scale and increasing the possible effect of each edit relative to the tolerance. By layer 19, the mean intervals are 1.19 and 1.38 for the naïve and Eq. 4 triggers. Remaining score slack, branch imbalance, and censoring also affect these means. This supports the observed depth trend without establishing a universal interval after every reevaluation.

Fewer checks do not by themselves imply less computation. At the root, 91.0% of naïve checks rescore no candidate-block entries, relying on cached bounds; the alternative refreshes the exact candidate minimum after a triggered decision. Both arms perform the same histogram-update work. The instrumented alternative takes 20.0 s for the update stream versus 11.8 s for the naïve trigger; the arms ran concurrently, so these times are descriptive rather than a controlled speed comparison. The experiment therefore establishes a change in decision frequency, not a production speedup or the benefit of deferring statistics updates into batches.