A APPENDIX

A.1 PSEUDOCODE

Algorithm 1 Training DYNFRS and Unlearning Samples with $\text{OCC}(q)$

procedure DISTRIBUTE(S, T, q)
  $k \leftarrow T \times q$
  $S^{(1)}, \ldots, S^{(T)} \leftarrow \varnothing, \cdots, \varnothing$
  for $\langle \mathbf{x}_i, y_i \rangle \in S$ **do**
    $j_{1 \cdots k} \leftarrow$ randomly and independently sample k different integers from $[T]$
    for $t \in j_{1 \cdots k}$ **do**
      $S^{(t)} \leftarrow S^{(t)} \cup \{\langle \mathbf{x}_i, y_i \rangle\}$
    end for
  end for
  return $S^{(1)}, \ldots, S^{(T)}$
end procedure

procedure TRAIN(S, T, q)
  $\Phi \leftarrow \varnothing$
  $S^{(1)}, \ldots, S^{(T)} \leftarrow$ DISTRIBUTE(S, T, q)
  for $t \leftarrow 1 \cdots T$ **do**
    $\varphi_t \leftarrow$ BUILDTREE($S^{(t)}$)
    $\Phi \leftarrow \Phi \cup \{\varphi_t\}$
  end for
  return Φ
end procedure

procedure ADD(Φ, $\langle \mathbf{x}, y \rangle$)
  $j_{1 \cdots k} \leftarrow$ randomly and independently sample k different integers from $[T]$
  for $t \leftarrow j_{1 \cdots k}$ **do**
    ADD($\varphi_t, \langle \mathbf{x}, y \rangle$)
  end for
end procedure

procedure REMOVE(Φ, $\langle \mathbf{x}, y \rangle$)
  for $t \leftarrow 1 \cdots T$ **do**
    if $\langle \mathbf{x}, y \rangle \in S^{(t)}$ **then**
      REMOVE($\varphi_t, \langle \mathbf{x}, y \rangle$)
    end if
  end for
end procedure

Algorithm 2 Unlearning and Querying in Trees with LZY

procedure REMOVE(φ, $\langle \mathbf{x}, y \rangle$) ▷ ADD is similar
  DELETE(ROOT(φ), $\langle \mathbf{x}, y \rangle$)
end procedure

procedure REMOVE(u, $\langle \mathbf{x}, y \rangle$)
  $S_u \leftarrow S_u \backslash \langle \mathbf{x}, y \rangle$ ▷ implementation of DYNFRS does not actually store S_u, so this line stands for updating all split statistics of node u
  if $\text{LZY}_u = 1$ **then return**
  else if BESTSPLITCHANGED(u) **then**
    $\text{LZY}_u \leftarrow 1$
  else if $\neg$ISLEAF(u) **then**
    if $\mathbf{x}_{a_u^\star} \leq w_u^\star$ **then**
      REMOVE($u_l, \langle \mathbf{x}, y \rangle$)
    else
      REMOVE($u_r, \langle \mathbf{x}, y \rangle$)
    end if
  end if
end procedure

procedure QUERY(φ, $\langle \mathbf{x}, y \rangle$)
  QUERY(ROOT(φ), $\langle \mathbf{x}, y \rangle$)
end procedure

procedure QUERY(u, $\langle \mathbf{x}, y \rangle$)
  if $\text{LZY}_u = 1$ **then**
    SPLIT(u)
    $\text{LZY}_u \leftarrow 0$
    $\text{LZY}_{u_l} \leftarrow 1, \text{LZY}_{u_r} \leftarrow 1$
  end if
  if ISLEAF(u) **then**
    return S_u
  end if
  if $\mathbf{x}_{a_u^\star} \leq w_u^\star$ **then**
    QUERY($u_l, \langle \mathbf{x}, y \rangle$)
  else
    QUERY($u_r, \langle \mathbf{x}, y \rangle$)
  end if
end procedure

A.2 PROOFS

Theorem 1. *Sample addition and removal for the* DYNFRS *framework are exact.*

Proof. We prove that the subsampling method $\text{OCC}(q)$ maintains the exactness of DYNFRS. Let random variable $o_{i,t} \triangleq [\langle \mathbf{x}_i, y_i \rangle \in S^{(t)}]$ denotes whether $\langle \mathbf{x}_i, y_i \rangle$ occurs in $S^{(t)}$. In $\text{OCC}(q)$, each sample $\langle \mathbf{x}_i, y_i \rangle$ is distributed to $\lceil qT \rceil$ distinct trees, with the selection of these trees being independent of other samples $\langle \mathbf{x}_j, y_j \rangle \in S$ for $j \neq i$. Thus, $o_{i,\cdot}$ is independent from $o_{j,\cdot}$ for $j \neq i$. However, $o_{i,1}, o_{i,2}, \ldots, o_{i,T}$ are dependent on each other, constrainted by $\forall t \in [T]$, $o_{i,t} \in \{0, 1\}$, $\sum_{t=0}^{T} o_{i,t} = \lceil qT \rceil$, and we say they follow a joint distribution $\mathcal{B}(T, q)$.

Now, let $S'^{(1)}, S'^{(2)}, \ldots, S'^{(T)}$ denotes the training sets for each tree generated by applying $\text{OCC}(q)$ to the modified training set S', and let $o'_{i,t} \triangleq [\langle \mathbf{x}_i, y_i \rangle \in S'^{(t)}]$. Then, when removing sample

$\langle \mathbf{x}_i, y_i \rangle$ (i.e., $S' = S \backslash \langle \mathbf{x}_i, y_i \rangle$), we have $(o_{j,1}, \ldots, o_{j,T}) \sim \mathcal{B}(T, q)$ and $(o'_{j,1}, \ldots, o'_{j,T}) \sim \mathcal{B}(T, q)$ for $i \neq j$, and $o_{i,\cdot} = 0$ (because $\langle \mathbf{x}_i, y_i \rangle$ is removed) and $o'_{i,\cdot} = 0$. Notably, $\mathcal{B}(T, q)$ depends on T and q only, but not on training samples S. This shows that simply setting $o_{i,\cdot} = 0$ ensures that $\textsc{occ}(q)$ with sample removed maintains the same distribution as applying $\textsc{occ}(q)$ on the modified training set.

Similarily, when adding $\langle \mathbf{x}_i, y_i \rangle$ (i.e., $S' = S \cup \langle \mathbf{x}_i, y_i \rangle$), $(o_{j,1}, \ldots, o_{j,T})$ and $(o'_{j,1}, \ldots, o'_{j,T})$ follow the same distribution $\mathcal{B}(T, q)$ for $j \neq i$. While $(o_{j,1}, \ldots, o_{j,T})$ is generated from $\mathcal{B}(T, q)$ for addition, it is equivalent to $(o'_{i,1}, \ldots, o'_{i,T})$ for following the same distribution. Therefore, $\textsc{occ}(q)$ with sample added maintains the same distribution as applying $\textsc{occ}(q)$ on the modified training set.

Next, we prove that the addition and removal operations are exact within a specific DYNFRS tree. When no change in range of attribute a ($[\min\{\mathbf{x}_{i,a} \mid \langle \mathbf{x}_i, y_i \rangle \in S_u\}, \max\{\mathbf{x}_{i,a} \mid \langle \mathbf{x}_i, y_i \rangle \in S_u\}]$) occurs, the candidate splits are sampled from the same uniform distribution making DYNFRS and the retraining method are identical in distribution node's split candidates. However, when a change in range occurs, DYNFRS resamples all candidate splits and makes them stay in the same uniform distribution as those in the retraining method. Consequently, DYNFRS adjusts itself to remain in the same distribution with the retraining method. Thus, sample addition and removal in DYNFRS are exact.

□

Lemma 1. *For a certain Extremely Randomized Tree node u, and a specific attribute a, the time complexity of finding the best split of attribute a is $\mathcal{O}(|S_u| \log s)$, assuming that $|S_u| \gg s$.*

Proof. Conventionally, for each tree node u and an attribute a, we uniformly samples s thresholds $w_{a,1\ldots s}$ from $[\min\{\mathbf{x}_{i,a} \mid \langle \mathbf{x}_i, y_i \rangle \in S_u\}, \max\{\mathbf{x}_{i,a} \mid \langle \mathbf{x}_i, y_i \rangle \in S_u\}]$. Then, we try to split S_u with each threshold and look for split statistics that are: (1) the number of samples in the left or right child (i.e., $|S_{u_l}|$ and $|S_{u_r}|$), and (2) the number of positive samples in left or right child ($|S_{u_l,+}|$ and $|S_{u_r,+}|$), which are the requirements for calculating the empirical criterion scores.

One approach, as used by prior works, first sort all samples S_u by $\mathbf{x}_{\cdot,a}$ in ascending order, and then sort thresholds $w_{a,1\ldots s}$ in ascending order. These sortings has a time complexity of $\mathcal{O}(|S_u| \log |S_u|)$ and $\mathcal{O}(s \log s)$, respectively. After that, a similar technique used in the merge sort algorithm is used to find the desired split statistics in $\mathcal{O}(|S_u| + s)$.

To get rid of the costly sorting on S_u, we sort $w_{a,1\ldots s}$ $\mathcal{O}(s \log s)$ and then iterate through all samples and calculate the changes each sample brings to candidates' split statistics. For convenience, let

$$b_k \triangleq \big|\{\langle \mathbf{x}_i, y_i \rangle \mid \langle \mathbf{x}_i, y_i \rangle \in S_u \wedge \mathbf{x}_{i,a} \leq w_{a,k}\}\big|,$$
$$c_k \triangleq \big|\{\langle \mathbf{x}_i, y_i \rangle \mid \langle \mathbf{x}_i, y_i \rangle \in S_u \wedge \mathbf{x}_{i,a} \leq w_{a,k} \wedge y_i = +\}\big|,$$

which are crucial split statistics for calculating the empirical criterion score. We start with setting $b_{1\ldots s}$ and $c_{1\ldots s}$ as all zeros. Then, for a sample $\langle \mathbf{x}_i, y_i \rangle \in S_u$, it will cause an increment in $b_{s'\ldots s}$ for some s' satisfying $\mathbf{x}_{i,a} \leq w_{s'}$ and $\mathbf{x}_{i,a} > w_{s'-1}$. Given that $w_{a,1\ldots s}$ are sorted, all k, $(s' \leq k \leq s)$ satisfy $\mathbf{x}_{i,a} \leq w_{a,k}$, while $\mathbf{x}_{i,a} > w_{a,k}$ for all $1 \leq k < s'$. s' can be easily found by binary search in $\mathcal{O}(\log s)$, then adding 1 to $b_{s'}, b_{s'+1}, \ldots, b_s$ is the only thing left. Use a loop for range addition is clearly $\mathcal{O}(s)$, but insteading of finding $b_{1\ldots s}$, we keep track of $d_{1\ldots s}$, where $d_k \triangleq b_k - b_{k-1}$. So increment $b_{s'\ldots s}$ can be replace by $d_{s'} \leftarrow d_{s'} + 1$, which is $\mathcal{O}(1)$. When all samples are processed, we construct $b_{1\cdot s}$ from $d_{1\ldots s}$, where prefix sums $b_k \leftarrow b_{k-1} + d_k$ help solve it in $\mathcal{O}(s)$.

For every sample $\langle \mathbf{x}_i, y_i \rangle \in S$, we need to find s' in $\mathcal{O}(\log s)$ (Algorithm 3: line 10), and perform increment in d'_s in $\mathcal{O}(1)$ (Algorithm 3: line 11), which results in a time complexity of $\mathcal{O}(|S_u| \log s)$ in this part (Algorithm 3: line 9-14). Meanwhile, the prefix sum is executed after all samples are processed (Algorithm 3: line 15-18), and its execution time is bounded by $\mathcal{O}(s)$. Luckily, $c_{1\ldots s}$ can be calculated in a similar manner, and with both $b_{1\ldots s}$ and $c_{1\ldots s}$ ready, we can obtain the empirical criterion score for each candidate split (Algorithm 3: line 21-28), and this has a time complexity of $\mathcal{O}(s)$ assuming calculating criterion scores to be $\mathcal{O}(1)$.

Since $|S_u| \gg s$, the term $\mathcal{O}(|S_u| \log s)$ dominates in time complexity, with the binary search (Algorithm 3: line 10) being the threshold. It is noteworthy that adopting exponential search to find s' can result in an expected $\mathcal{O}(\log \log s)$ time complexity since $w_{a,1\ldots s}$ are uniformly distributed.

However, binary search outperforms exponential search in practice, so we conclude with a time complexity of $\mathcal{O}(|S_u| \log s)$ for finding the best split of attribute a in node u when $|S_u| \gg s$.

Algorithm 3 Find the best split for attribute a

procedure FINDATTRIBUTEBESTSPLIT(S_u, a, s)
  $R \leftarrow [\min\{\mathbf{x}_{i,a} \mid \langle \mathbf{x}_i, y_i \rangle \in S_u\}, \max\{\mathbf{x}_{i,a} \mid \langle \mathbf{x}_i, y_i \rangle \in S_u\}]$
  $w_{a,1\cdots s} \leftarrow$ sample s i.i.d. values from R
  $w_{a,1\cdots s} \leftarrow \text{SORT}(w_{a,1\cdots s})$ ▷ sort $w_{a,1\cdots s}$ in ascending order.
  $b_{1\cdots s} \leftarrow \{0, \cdots, 0\}$ ▷ create an 1-D array of size s
  $c_{1\cdots s} \leftarrow \{0, \cdots, 0\}$ ▷ create an 1-D array of size s
  $n \leftarrow |S_u|$
  $n_+ \leftarrow 0$
  for $\langle \mathbf{x}_i, y_i \rangle \in S_u$ **do**
    $s' \leftarrow \text{LOWERBOUND}(w_{1\cdots s}, \mathbf{x}_{i,a})$ ▷ find the largest position k s.t. $w_{k-1} < \mathbf{x}_{i,a}$
    $b_{s'} \leftarrow b_{s'} + 1$
    $c_{s'} \leftarrow c_{s'} + y_i$
    $n_+ \leftarrow n_+ + y_i$
  end for
  for $k \leftarrow 2 \ldots s$ **do** ▷ find prefix sum
    $b_k \leftarrow b_k + b_{k-1}$
    $c_k \leftarrow c_k + c_{k-1}$
  end for
  $I^\star \leftarrow \infty$
  $w_a^\star \leftarrow 0$
  for $k \leftarrow 1 \ldots s$ **do**
    $I_k \leftarrow I(b_k, c_k, n - b_k, n_+ - c_k)$
    ▷ $I(|S_{u_l}|, |S_{u_l,+}|, |S_{u_r}|, |S_{u_r,+}|)$ returns the empirical critierion score for split $(a, w_{a,k})$ in $\mathcal{O}(1)$ using either Gini index I_G (this work) or Shannon's entropy I_E.
    if $I_k < I^\star$ **then**
      $I^\star \leftarrow I_k$
      $w_a^\star \leftarrow w_{a,k}$
    end if
  end for
  return $(a, w_a^\star)$
end procedure

□

Given $q < 1$, the proportion of trees each sample is assigned to, T, the number of trees in the forest, $d_{\max}$, the maximum depth of each tree, p, the number of candidate attributes, s, the number of candidate splits for each attribute (usually $s \leq 30$), and $n = |S|$, size of the training set, we now prove the following:

Theorem 2. *Training* DYNFRS *yields a time complexity of* $\mathcal{O}(qTd_{\max}pn \log s)$.

Proof. For certain tree and a specific node u, we find the best split among p randomly selected attributes $a_{1\cdots p}$, and we call FINDATTRIBUTEBESTSPLIT(S_u, a_i, s) (Algorithm 3) p times for each $i \in [p]$. From Lemma 1, finding the best split for the node u has a time complexity of $\mathcal{O}(p|S_u| \log s)$. Then, summing $|S_u|$ over all tree nodes u on that tree, we have $\sum_u |S_u| \leq d_{\max} qn$, since the root of the tree contains about $\lceil qT \rceil n/T \approx qn$ samples, and each layer has at most the same amount of samples as the root (layer 0). Therefore, the time complexity for training one DYNFRS tree can be bounded by $\mathcal{O}(d_{\max} qn \log s)$. Since there are T independent trees in the forest, the time complexity for training a DYNFRS forest is $\mathcal{O}(qTd_{\max}pn \log s)$.

□

Theorem 3. *Modification (sample addition or removal) in* DYNFRS *yields a time complexity of* $\mathcal{O}(qTd_{\max}ps)$ *if no attribute range changes occurs while* $\mathcal{O}(qTd_{\max}ps + cn_{\mathit{aff}} \log s)$ *otherwise (where* c *denotes the number of attributes affected, and* n_{aff} *denotes the sum of sample size among all affected nodes met by this modification request).*

Proof. When no attribute range change occurs on each tree, the modification request traverses a path from the root to a leaf with at most $d_{\max}$ nodes. For each node, we need to recalculate all the empirical criterion scores for all candidate splits $\mathcal{O}(ps)$. Since $\textsc{occ}(q)$ guarantees that only $\lceil qT \rceil$ trees are affected by the modification requests, at most $qTd_{\max}$ nodes need the recalculation. So, the time complexity for one modification request yields $\mathcal{O}(qTd_{\max}ps)$.

When an attribute range occurs on u, it is necessary to call $\textsc{FindAttributeBestSplit}(S_u, a, s)$ for u and the affected attribute a. Given that the affected nodes' sample sizes sum up to n_{aff}, and for each affected node, we need to resample at most $c \leq p$ attributes, and then Lemma 1 entails that the time complexity for completing all resampling is an additional $\mathcal{O}(cn_{\text{aff}} \log s)$.

□

Theorem 4. *Query in* DynFrs *yields a time complexity of* $\mathcal{O}(Td_{\max})$ *if no lazy tag is met, while* $\mathcal{O}(Td_{\max} + pn_{lzy} \log s)$ *otherwise (where* n_{lzy} *denotes the sum of sample size among all nodes with lazy tag and met by this query).*

Proof. On each tree, the query starts with the root and ends at a leaf node, traversing a tree path with at most $d_{\max}$ nodes, and the query on DynFrs aggregates the results of all T trees, therefore querying without bumping into a lazy tag yields a time complexity of $\mathcal{O}(Td_{\max})$.

However, if the query reaches on a tagged node u, we need to perform a split on it, and by the proof of Theorem 2 and Lemma 1, finding the best split of node u calls function $\textsc{FindAttributeBestSplit}(S_u, \cdot, s)$ p times and results in a time complexity of $\mathcal{O}(p|S_u| \log s)$. As n_{lzy} denotes the sum of sample sizes of all nodes with lazy tags met by the query, handling these lazy tags requires an additional time complexity of $\mathcal{O}(pn_{\text{lzy}} \log s)$.

□

A.3 Implementation

All of the experiments are conducted on a machine with AMD EPYC 9754 128-core CPU and 512 GB RAM in a Linux environment (Ubuntu 22.04.4 LTS), and all codes of DynFrs are written in C++ and compiled with the g++ 11.4.0 compiler and the -O3 optimization flag enabled. To guarantee fair comparison, all tests are run on a single thread and are repeated 5 times with the mean and standard deviation reported.

DynFrs is tuned using 5-fold cross-validation for each dataset, and the following hyperparameters are tuned using a grid search: Number of trees in the forest $T \in \{100, 150, 250\}$, maximum depth of each tree $d_{\max} \in \{10, 15, 20, 25, 30, 40\}$, and the number of sampled splits $s \in \{5, 15, 20, 30, 40\}$.

A.4 Baselines

HedgeCut and OnlineBoosting can not process real continuous input. Thus, all numerical attributes are discretized into 16 bins, as suggested in their works (Schelter et al., 2021; Lin et al., 2023). Both of them are not capable of processing samples with sparse attributes, so one-hot encoding is disabled for them. Additionally, it is impossible to train Hedgecut on datasets Synthetic and Higgs in our setting due to its implementation issue, as its complexity degenerates to $\mathcal{O}(pn^2)$ sometimes and consumes more than 256 GB RAM during training.

A.5 Datasets

Purchase (Sakar and Kastro, 2018; Dua and Graff, 2019) is primarily used to predict online shopping intentions, i.e., users' determination to complete a transaction. The dataset was collected from an online bookstore built on an osCommerce platform.

Vaccine (Bull et al., 2016; DrivenData, 2019) comes from data-mining competition in DrivenData. It contains 26,707 survey responses, which were collected between October 2009 and June 2010. The survey asked 36 behavioral and personal questions. We aim to determine whether a person received a seasonal flu vaccine.

Adult (Becker and Kohavi, 1996; Dua and Graff, 2019) is extracted from the 1994 Census database by Barry Becker, and is used for predicting whether someone's income level is more than 50,000 dollars per year or not.

Bank (Moro et al., 2014; Dua and Graff, 2019) is related to direct marketing campaigns of a Portuguese banking institution dated from May 2008 to November 2010. The goal is to predict if the client will subscribe to a term deposit based on phone surveys.

Heart (Kaggle, 2018) is provided by Ulianova, and contains 70,000 patient records about cardiovascular diseases, with the label denoting the presence of heart disease.

Diabetes (Strack et al., 2014; Dua and Graff, 2019) encompasses a decade (1999-2008) of clinical diabetes records from 130 hospitals across the U.S., covering laboratory results, medications, and hospital stays. The goal is to predict whether a patient will be readmitted within 30 days of discharge.

Synthetic (Kaggle, 2016) focuses on the patient's appointment information, such as date, number of SMS sent, and alcoholism, aiming to predict whether the patient will show up after making an appointment.

Higgs (Baldi et al., 2014; Dua and Graff, 2019) consists of 1.1×10^7 signals characterized by 22 kinematic properties measured by detectors in a particle accelerator and 7 derived attributes. The goal is to distinguish between a background signal and a Higgs boson signal.

A.6 RESULTS

In this section, Table 3 presents the training time for each model, with OnlineBoosting being the fastest in most datasets while DYNFRS ranks first among Random Forest based methods. Table 4, 5, 6, and 7 despicts the runtime for model simultaneously unlearning 1, 10, 100 instances or 0.1% and 1% of all samples, where DYNFRS consistently outperforms all others in all settings and all datasets.

Table 3: Training time (↓) of each model, measured in seconds (s), and the standard deviation is given with the same unit in a smaller font. "/" means the model is unable to train on that dataset.

Datasets	DaRE	HedgeCut	Online Boosting	DYNFRS ($q = 0.1$)	DYNFRS ($q = 0.2$)
Purchase	3.10±0.0	1.05±0.0	**0.27**±0.0	0.38±0.0	0.72±0.0
Vaccine	4.78±0.0	431±14	**1.05**±0.0	1.12±0.0	2.27±0.0
Adult	5.02±0.1	11.8±0.5	0.77±0.0	**0.61**±0.0	1.15±0.0
Bank	8.26±0.2	8.44±0.3	**0.92**±0.0	1.15±0.0	2.37±0.0
Heart	12.1±0.2	3.51±0.0	**1.02**±0.0	1.04±0.0	1.96±0.0
Diabetes	123±1.0	162±3.3	**3.51**±0.0	8.67±0.0	18.2±0.0
NoShow	65.4±0.4	28.1±0.3	**1.68**±0.0	3.08±0.0	6.10±0.0
Synthetic	1334±6.3	/	**40.7**±0.9	66.3±0.2	128±0.4
Higgs	10793±48	/	**460**±13	548±1.2	1120±1.0

Table 4: Runtime (↓) for each model to unlearn 1 sample measured in milliseconds (ms), and the standard deviation is given with the same unit in a smaller font. "/" means the model is unable to train on that dataset or unlearning takes too long.

Datasets	DaRE	HedgeCut	Online Boosting	DYNFRS
Purchase	35.0±15	1245±343	83.4±9.0	**0.40**±0.2
Vaccine	16.0±15	33445±14372	222±53	**1.40**±0.9
Adult	10.6±5.0	3596±2396	249±62	**1.10**±1.5
Bank	33.2±17	2760±371	227±7.4	**2.40**±3.7
Heart	16.8±10	972±154	411±13	**0.50**±0.2
Diabetes	293±168	27654±10969	753±143	**7.30**±5.4
NoShow	330±176	1243±94	570±69	**0.30**±0.0
Synthetic	2265±3523	/	5225±241	**2.50**±3.7
Higgs	174±135	/	73832±4155	**1.60**±1.8

Table 5: Runtime (↓) for each model to unlearn 10 samples measured in milliseconds (ms), and the standard deviation is given with the same unit in a smaller font. "/" means the model is unable to train on that dataset or unlearning takes too long.

Datasets	DaRE	HedgeCut	Online Boosting	DYNFRS
Purchase	295±54	10973±643	183±21	**12.3**±4.8
Vaccine	285±160	222333±64756	418±41	**9.20**±2.8
Adult	148±79	51831±2795	389±48	**4.80**±2.4
Bank	320±108	18091±1524	423±47	**7.6**±2.7
Heart	162±46	5524±335	625±36	**6.80** ±2.8
Diabetes	2773±877	211640±79652	1096±172	**85.5**±38
NoShow	2217±723	17235±17905	712±74	**18.2**±123
Synthetic	92279±42634	/	6015±301	**77.3**±34
Higgs	20119±31897	/	104063±1258	**32.1**±8.6

Table 6: Runtime (↓) for each model to unlearn 100 samples measured in milliseconds (ms), and the standard deviation is given with the same unit in a smaller font. "/" means the model is unable to train on that dataset or unlearning takes too long.

Datasets	DaRE	HedgeCut	Online Boosting	DYNFRS
Purchase	3591±186	83649±2047	275±12	**70.4**±11
Vaccine	2385 ±408	1703355±157274	792±15.06	**82.6**±11
Adult	954±158	219392±34630	632±24	**32.2**±4.0
Bank	3546±302	195014±15854	740±20	**78.8**±13
Heart	1502±543	33806±5385	986±75	**59.3**±11
Diabetes	23833±10330	/	2071±115	**578**±50
NoShow	23856±3978	57021±6327	1120±92	**117**±10
Synthetic	1073356±420053	/	7609±171	**889**±287
Higgs	165122±149092	/	145386±5677	**642**±317

Table 7: Left: runtime (↓) for unlearning 0.1% of the training set between models. Right: runtime (↓) for unlearning 1% of the training set between models. Each cell contains the elapsed time in seconds (s), and the standard deviation is given with the same unit in a smaller font. "/" means the model is unable to train on that dataset or unlearning takes too long.

Datasets	DaRE	HedgeCut	Online Boosting	DYNFRS	DaRE	HedgeCut	Online Boosting	DYNFRS
Purchase	0.35±0.1	11.25±1.5	0.17±0.0	**0.01**±0.0	3.39±0.7	76.0±3.0	0.28±0.0	**0.07**±0.0
Vaccine	0.47±0.1	404.73±69	0.61±0.1	**0.02**±0.0	5.01±1.0	4054±427	0.98±0.0	**0.13**±0.0
Adult	0.44±0.3	88.1±27	0.49±0.1	**0.01**±0.0	3.39±0.6	516±23	0.80±0.0	**0.09**±0.0
Bank	1.15±0.3	47.3±6.0	0.61±0.1	**0.02**±0.0	14.7±2.3	418±25	0.96±0.0	**0.16**±0.0
Heart	0.70±0.2	20.0±1.7	0.85±0.0	**0.03**±0.0	8.43±1.2	145±10	1.23±0.0	**0.20**±0.0
Diabetes	23.8±2.7	694±61	2.12±0.1	**0.57**±0.1	258±20	/	3.51±0.1	**2.50**±0.1
NoShow	18.8±2.7	57.0±6.3	1.10±0.1	**0.10**±0.0	268±7.5	/	1.90±0.1	**0.56**±0.0
Synthetic	10790±5348	/	13.1±0.6	**5.68**±0.2	/	/	44.2±1.4	**27.4**±0.9
Higgs	/	/	188±7.1	**39.2**±0.7	/	/	456±4.7	**201**±9.6

A.7 SPACE COMPLEXITY AND MEMORY CONSUMPTION

The space complexity of DYNFRS is $\mathcal{O}(qTn + Tvps)$ where T is the number of trees in the forest, n is the number of samples in the training sets, q is the factor used in OCC, v is the average number of nodes in each tree, p is the number of attributes considered by each node, and s is the number of candidate splits. Since we store training samples on each leaf, and each tree occupies qn samples on

average, then the leaf statistics sums up to $\mathcal{O}(qTn)$. As mentioned in section 4.3, we store an extra $\mathcal{O}(ps)$ split statistics on each node so that these split statistics contribute up to $\mathcal{O}(Tvps)$ space.

We compare the maximum resident space of DYNFRS and DaRE for building on the training set. We found that DaRE, which has a space complexity of $\mathcal{O}(Tn+Tvps)$, has larger memory consumption than DYNFRS in all datasets. We use `/usr/bin/time` in Linux to evaluate the maximum resident set size.

Table 8: The maximum resident set size (↓) of each model measured in megabytes with standard deviation shown in a smaller font.

Datasets	DaRE	DYNFRS
Purchase	398.4±1.4	**83.2**±1.6
Vaccine	782.2±1.6	**492.8**±1.6
Adult	460.6±2.2	**232.0**±0
Bank	801.0±1.8	**300.0**±0
Heart	254.4±1.7	**252.2**±1.6
Diabetes	7148±39	**2512**±7.6
NoShow	3792±19	**761.6**±2.0
Synthetic	8526±65	**5827**±7.8
Higgs	60528±789	**58263**±52

A.8 MULTI-CLASS CLASSIFICATION

DYNFRS is capable of handling multi-class classification tasks since OCC(q) and LZY do not affect the functionality of the forest. We tested DYNFRS's prediction performance and unlearning boost on 3 datasets — Optical (Hyafil and Rivest, 1976), Pen (Alpaydin and Alimoglu, 1996), and Letter (Slate, 1991). Unfortunately, existing random forest unlearning methods (DaRE and HedgeCut) have no implementation for multi-class classification, so we only include the Random Forest Classifier implementation (we call it Vanilla in the following) in scikit-learn as our baseline.

Table 9: Datasets specifications. (# train: number of training samples; # test: number of testing samples; # attr: number of attributes; # class: number of label classes.)

Datasets	# train	# test	# attr	# class
Optical	3823	1797	64	10
Pen	7494	3498	16	10
Letter	15000	5000	16	26

Table 10: Each model's predictive performance, training time, and unlearning boost with standard deviation shown in a smaller font.

	Accuracy (↑)		Train Time (↓,ms)		Unlearn Boost (↑)	
Datasets	Vanilla	DYNFRS	Vanilla	DYNFRS	Vanilla	DYNFRS
Optical	.9694±.002	**.9707**±.002	585.4±1.9	**445.4**±2.7	1±0	**292.5**±12.1
Pen	.9649±.002	**.9696**±.002	996.2±1.7	**813.2**±8.0	1±0	**603.0**±44.8
Letter	.9603±.002	**.9624**±.001	1666±7.6	**1530**±20	1±0	**1006**±54.2

Results show that DYNFRS outperforms the vanilla Random Forest in terms of predictive performance and training time for all datasets. Additionally, DYNFRS still shows splendid unlearning efficiency on these datasets. In all datasets, DYNFRS outperforms Vanilla by 2-3 order of magnitudes in terms of unlearning efficiency. Also notice that DYNFRS is poorly optimized for multi-class classification tasks.

In the multi-class classification setting, we suggest picking $q = 0.5$ for OCC(q), because the rise in class number leads to the drop of sample size to each class (roughly n/c, where n is the number of training samples and c is the number of classes) and rising q enable each tree is accessible to more samples belonging to a specific class and lead to better predictive performance eventually. For all datasets, we set the hyperparameters of DYNFRS to $T = 100$, $d_{\max} = 20$, and $s = 1$.

A.9 REGRESSION

Regression tasks are not implemented and evaluated by any of the existing tree-based unlearning methods. Since there is no such baseline, we compare DYNFRS's predictive performance (via mean squared error) and unlearning efficiency (via unlearning boost) against the naive retraining method. We used a large and popular dataset Bike (Fanaee-T, 2013) for the regression task.

Table 11: Datasets specifications. (# train: number of training samples; # test: number of testing samples; # attr: number of attributes.)

Datasets	# train	# test	# attr
Bike	13903	3476	16

Table 12: Each model's predictive performance, training time, and unlearning boost with standard deviation are shown in a smaller font.

	MSE (↓)		Train Time (↓,ms)		Unlearn Boost (↑)	
Datasets	Vanilla	DYNFRS	Vanilla	DYNFRS	Vanilla	DYNFRS
Bike	$3.904_{\pm.071}$	$\mathbf{3.011}_{\pm.167}$	$6796_{\pm 6.5}$	$\mathbf{3412}_{\pm 54}$	$1_{\pm 0}$	$\mathbf{2368}_{\pm 118}$

Results show that DYNFRS outperforms the vanilla Random Forest in terms of predictive performance and training time. Note that the implementation of DYNFRS regressor is rough and poorly optimized due to short time slots. However, DYNFRS still shows outstanding unlearning efficiency on regression task, as DYNFRS achieves an averaged 2368 unlearning boost in dataset Bike.

In the regression setting, we suggest picking $q = 0.5$ for lower mean squared error. In the experiment, we set the hyperparameters of DYNFRS to $T = 100$, $d_{\max} = 15$, and $s = 5$.

A.10 ONLINE MIXED DATA STREAM

Table 13: DYNFRS's latency (↓) of sample addition/removal and querying in 4 scenarios measured in microseconds (μs). # add/del/qry: the number of sample addition/removal/querying requests; add/del/qry lat.: the latency of sample addition/removal/querying. Each cell contains the averaged latency with its minimum and maximum values listed in a smaller font.

No.	# add	# del	# qry	add lat. (↓, μs)	del lat. (↓, μs)	qry lat. (↓, μs)
1	$5 \cdot 10^5$	$5 \cdot 10^5$	10^6	406.2 [150, 450894]	437.6 [134, 394801]	3680 [209, 1885030]
2	$5 \cdot 10^5$	$5 \cdot 10^5$	10^6	122.7 [30, 168984]	120.2 [25, 146300]	1218 [23, 1026964]
3	$5 \cdot 10^4$	$5 \cdot 10^4$	10^6	140.0 [30, 81661]	139.2 [32, 101429]	299.5 [21, 861151]
4	$5 \cdot 10^3$	$5 \cdot 10^3$	10^6	145.5 [44, 55954]	140.5 [38, 29810]	72.3 [19, 212915]

To simulate a large-scale database, we use the Higgs dataset, the largest in our study. We train DYNFRS on $88,000,000$ samples and feed it with mixed data streams with different proportions of modification requests. Scenario 1 is the vanilla single-thread setting, while scenarios 2, 3, and 4 employ 25 threads using OpenMP. DYNFRS achieves an averaged latency of less than 0.15 ms for modification requests (Table 13 column # add and # del) and significantly outperforms DaRE, which requires 180 ms to unlearn a single instance on average. Query latency drops from 1.2 ms to 0.07 ms as the number of modification requests declines, as fewer lazy tags are introduced to trees.

These results are striking: while it takes over an hour to train a vanilla Random Forest on Higgs, DYNFRS maintains exceptionally low latency that is measured in μs, even in the single-threaded setting. This makes DYNFRS highly suited for real-world scenarios, especially when querying constitutes a large proportion of requests (Table 13 Scenario 4).

A.11 EFFECTS ON NUMBER OF CANDIDATES

We assessed the predictive performance of DYNFRS with differnet candidate number s. We tested DYNFRS($q = 0.1$) with $s \in \{1, 5, 10, 20, 30, 50\}$ and report the predictive performance on all the binary classification datasets (same settings as in Section 5.2). The results are summarized in the table below:

Table 14: The predictive performance (↑) of DYNFRS($q = 0.1$) with $s \in \{1, 5, 10, 20, 30, 50\}$.

Datasets	$s = 1$	$s = 5$	$s = 10$	$s = 20$	$s = 30$	$s = 50$
Purchase	.9242$\pm$.001	.9313$\pm$.000	.9323$\pm$.000	.9329$\pm$.001	.9328$\pm$.001	.9330$\pm$.001
Vaccine	.7911$\pm$.002	.7910$\pm$.000	.7908$\pm$.001	.7910$\pm$.002	.7911$\pm$.002	.7912$\pm$.001
Adult	.8558$\pm$.001	.8610$\pm$.001	.8624$\pm$.001	.8635$\pm$.001	.8635$\pm$.000	.8640$\pm$.001
Bank	.9323$\pm$.000	.9399$\pm$.000	.9409$\pm$.000	.9414$\pm$.001	.9417$\pm$.000	.9417$\pm$.001
Heart	.7365$\pm$.001	.7359$\pm$.001	.7357$\pm$.001	.7359$\pm$.002	.7357$\pm$.000	.7351$\pm$.001
Diabetes	.6429$\pm$.001	.6453$\pm$.001	.6451$\pm$.001	.6446$\pm$.001	.6442$\pm$.001	.6443$\pm$.001
NoShow	.7278$\pm$.001	.7332$\pm$.000	.7328$\pm$.000	.7332$\pm$.000	.7323$\pm$.001	.7328$\pm$.000
Synthetic	.9352$\pm$.000	.9415$\pm$.000	.9421$\pm$.000	.9422$\pm$.000	.9424$\pm$.000	.9423$\pm$.000
Higgs	.7277$\pm$.000	.7409$\pm$.000	.7423$\pm$.000	.7431$\pm$.000	.7431$\pm$.000	.7431$\pm$.000

From the result, we find that the performance peaks around $s = 20$ in most datasets, and the result of $s = 30$ and $s = 50$ has no significant difference. However, in datasets Heart, Diabetes, and NoShow, a smaller s has an even higher predictive performance, suggesting that considering more candidates might not always be the best choice. These results indicate that the optimal s is dataset-specific and can be tuned for improved predictive performance in DYNFRS.

A.12 HYPERPARAMETERS

All hyperparameters of DYNFRS are listed in Table 15. Specially, we set the minimum split size of each node to be 10 for all datasets.

Table 15: Hyperparameters used by DYNFRS with both $q = 0.1$ and $q = 0.2$ setting.

Datasets	T	$d_{\max}$	s
Purchase	250	10	30
Vaccine	250	20	5
Adult	100	20	30
Bank	250	25	30
Heart	150	15	5
Diabetes	250	30	5
NoShow	250	20	5
Synthetic	150	40	30
Higgs	100	30	20