DFDT: Dynamic Fast Decision Tree for IoT Data Stream Mining on Edge Devices

Afonso Lourenço[1], João Rodrigo[1], João Gama[2], Goreti Marreiros[1]

[1]GECAD, ISEP, Polytechnic of Porto, Rua Dr. António Bernardino de Almeida, Porto, 4249-015, Portugal
[2]INESC-TEC, FEP, University of Porto, Rua Dr. Roberto Frias, Porto, 4200-465, Portugal

Abstract

The Internet of Things generates massive data streams, with edge computing emerging as a key enabler for online IoT applications and 5G networks. Edge solutions facilitate real-time machine learning inference, but also require continuous adaptation to concept drifts. While extensions of the Very Fast Decision Tree (VFDT) remain state-of-the-art for tabular stream mining, their unregulated growth limit efficiency, particularly in ensemble settings where post-pruning at the individual tree level is seldom applied. This paper presents DFDT, a novel memory-constrained algorithm for online learning. DFDT employs activity-aware pre-pruning, dynamically adjusting splitting criteria based on leaf node activity: low-activity nodes are deactivated to conserve resources, moderately active nodes split under stricter conditions, and highly active nodes leverage a skipping mechanism for accelerated growth. Additionally, adaptive grace periods and tie thresholds allow DFDT to modulate splitting decisions based on observed data variability, enhancing the accuracy–memory–runtime trade-off while minimizing the need for hyperparameter tuning. An ablation study reveals three DFDT variants suited to different resource profiles. Fully compatible with existing ensemble frameworks, DFDT provides a drop-in alternative to standard VFDT-based learners.

Introduction

The Internet of Things (IoT) connects a vast network of physical devices, generating massive, high-speed data streams (Gaber et al. 2014). To extract valuable insights from this ever-growing stream, the edge computing paradigm has emerged as a key enabler for low-latency IoT applications and 5G networks, bridging the gap between cloud services and end-users (Lourenço et al. 2025b). However, benefiting from edge solutions not only implies bringing machine learning models for real-time inference, but also continuous updates to adapt to the evolving nature of unbounded streams. Unlike batch learning, where all training data is available upfront, data streams arrive incrementally (Gama et al. 2010; Lourenço et al. 2025a). Thus, making models susceptible to becoming outdated, especially due to concept drifts, i.e., changes in data distribution over time.

To circumvent these challenges, many stream mining algorithms have been proposed, with extensions of the Very Fast Decision Tree (VFDT) (Domingos and Hulten 2000) being the state-of-the-art for tabular data sources. Their success can be attributed to approximation-based splitting, with incrementally updated statistical summaries of entropy-based metrics, e.g. information gain, to determine whether the observed utility of a split is statistically significant when the data distribution is unknown, e.g. via the Hoeffding bound (HB). This enables trees to grow incrementally, refining their partitions of the input space as more data arrives. However, uncontrolled tree growth can lead to excessive memory usage and reduced adaptability.

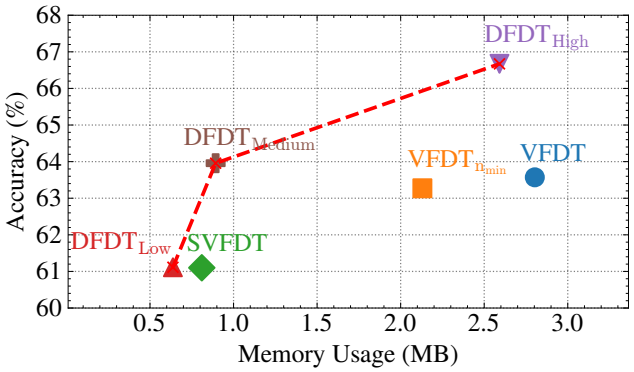

Figure 1: *Accuracy* x *Memory*

To counter this, adaptive tree algorithms introduce post-pruning mechanisms for forgetting. For instance, CVFDT (Hulten, Spencer, and Domingos 2001) evaluates alternate subtrees in parallel, replacing them when they outperform their predecessors. HAT (Bifet and Gavalda 2009) generalizes this with multiple recursive replacements, while EFDT (Manapragada, Webb, and Salehi 2018) aggressively prunes subtrees when better local structures emerge. Other methods, like UFFT (Gama, Medas, and Rodrigues 2005) and OnlineTree2 (Núñez, Fidalgo, and Morales 2007), use a short-term memory to stage and evaluate candidate subtrees, enabling smoother transitions and compact models.

Despite these innovations, single decision trees often suffer from instability and limited predictive performance. As a result, the community has largely shifted towards ensemble-based methods, where multiple diverse trees are combined

to enhance robustness and adaptability (Gomes et al. 2017; Krawczyk et al. 2017). Ensembles handle concept drift at a higher level, dynamically updating individual components based on performance signals and using drift detectors to add, remove, or reweight learners (Neves et al. 2025). Thus, reducing the need for post-pruning at the individual tree level. As a result, most state-of-the-art ensembles still adopt the original VFDT as their base learner, often leading to inefficient memory use and redundant computation. Since post-pruning is rarely applied in ensemble settings, this raises a critical question: can pre-pruning strategies be leveraged to proactively regulate VFDT's growth, preventing unnecessary expansion in uncertain or low-utility regions of the input space?

To this end, a novel algorithm tailored for memory-constrained data stream mining is introduced, coined as DFDT: Dynamic Fast Decision Tree, whose main contributions are:

- different splitting conditions according to leaf activity, so that the algorithm can grow more organically depending on the distribution and number of instances observed at each leaf of the tree:
  - for nodes with low activity, DFDT deactivates the leaf node, conserving memory and computational resources.
  - for nodes with moderate activity, DFDT applies conservative splitting constraints based on entropy, information gain, and the number of instances observed at the leaf, ensuring controlled growth.
  - for highly active nodes, DFDT potentially employs a skipping mechanism, enabling rapid growth in response to significant data changes.
- dynamically adjustable grace periods and tie thresholds, allowing the algorithm to either delay or accelerate splits depending on data variability. This flexibility improves the accuracy–memory–runtime trade-off and reduces the need for extensive hyperparameter tuning.

An ablation study of these DFDT's components reveals three competitive variants tailored to different accuracy–memory trade-offs, summarized in Figure 1. These variants are fully compatible with existing ensemble frameworks (Gomes et al. 2017; Krawczyk et al. 2017), offering a drop-in alternative to standard VFDT-based learners.

Related Work

The core component for incrementally constructing decision trees (DTs) on edge devices, while maintaining a fixed time complexity per sample, is approximation-based splitting (Domingos and Hulten 2000). As new instances arrive, they are processed from the root to a leaf node, updating statistics at each node. These updates are used to periodically adjust heuristic values for each attribute, such as information gain (IG) and Gini index (GI) (Domingos and Hulten 2000). The tree attempts to perform splits based on a statistical bound that quantifies the confidence interval for the heuristic function, given a minimum amount of data. Typically, DTs compare the Hoeffding bound (HB) to the difference in evaluation between the best and second-best attribute splits (Domingos and Hulten 2000). When this difference exceeds the bound, the leaf node is split into child nodes.

Splitting rules. While traditional incremental DTs exclusively focus on evaluating the top two attributes, one can introduce extra splitting rules to promote even more valuable splits. For example, M-VFDT (Yang and Fong 2011) incorporates the fluctuation of the HB, tracking its mean, minimum, and maximum values, along with an accuracy metric, as a pre-pruning condition for split decisions. Alternatively, SVFDT (da Costa, de Leon Ferreira, and Duarte 2018) applies constraints that compare current metrics against historical data and cross-leaf information, while introducing a skipping mechanism to bypass these constraints when significant changes in the data are detected. Furthermore, REG-VFDT (Barddal and Enembreck 2020) adds a constraint to assess whether the best-ranked feature, chosen for splitting at a leaf node, provides substantial gains compared to the gains observed in previous splits within the same branch.

Tie breaking threshold (τ). A critical aspect of incremental DTs is the tie-breaking procedure. While using the statistical difference in IG between two attributes helps control tree growth, competition between two equally favored split candidates can hinder progress, especially when either option would be equally suitable. To mitigate this, if the difference in heuristic values exceeds a predefined tie threshold, denoted as τ, the split is performed. This threshold effectively controls the minimum rate of tree growth, with the attributes' ability to separate before reaching the threshold influencing the speed at which the tree expands. However, a fixed τ value may cause ties to be broken prematurely, before a meaningful decision can be made, due to a lack of suitable candidates rather than a true tie situation. To address this, VFDT-τ (Holmes, Richard, and Pfahringer 2005) compares the difference between the best and second-best candidates with the difference between the second-best and worst candidates, while M-VFDT (Yang and Fong 2011) designs an adaptive tie threshold that is dynamically calculated from the mean of HB, which has been shown to be proportionally related to the input stream samples.

Grace period ($n_{\min}$). To prevent premature splits with small sample sizes that undermine the validity of the HB, a grace period $n_{\min}$ specifies the minimum number of instances a leaf must observe before considering a split. However, without leveraging any information from the processed data, this can still result in computationally expensive split attempts or unnecessary delays in predictions. An alternative approach involves detecting frequent tie-breaking situations and applying steps to reduce the frequency of ties. For example, the default tie-breaking wait period can be adjusted by increasing the wait period for the child nodes after each tie is broken. This effect is cumulative until a valid split is found, after which the wait period is reset to the default value (Holmes, Richard, and Pfahringer 2005). Additionally, local statistics can be used to predict the optimal split time, minimizing delays and unnecessary split attempts. For instance, OSM (Losing, Wersing, and Hammer 2018) uses class dis-

tributions from previous split attempts to estimate the minimum number of examples required before the HB. Alternatively, VFDT-$n_{\min}$ (García-Martín et al. 2018) adjusts the grace period after unsuccessful split attempts, according to the reason of failure in order to ensure a split in the next iteration. For example, if the best attributes are not too similar, but their IG difference is insufficient to trigger a split, the solution is to wait for additional examples until the HB decreases enough to be smaller than the IG difference, adjusting the $n_{\min}$ accordingly. Similarly, if the top attributes are very similar in terms of IG, but the tie threshold τ is not exceeded, more instances are needed to allow the HB to decrease below τ, with $n_{\min}$ adjusted based on their IG difference (García-Martín et al. 2018).

Leaf activity. Another memory-efficient strategy is to condition split decisions based on leaf activity. GAHT (Garcia-Martin et al. 2022) devises a normalized measure to quantify how many instances have been observed at a particular leaf relative to the mean number of instances per leaf since its creation. Based on this value, nodes can either be deactivated when their activity is low, halting further splits to conserve resources, or more aggressive expansion strategies can be applied when activity exceeds a threshold, accelerating the growth of important nodes by relaxing the splitting rule to a less strict alternative. These adaptive expansion modes offer a more nuanced approach to tree growth, ensuring computational resources are focused on expanding nodes that significantly influence the model's accuracy, while avoiding unnecessary splits in less relevant parts of the tree. Alternatively, VFDT (Domingos and Hulten 2000) deactivates the least promising leaves based on the probability that examples will reach those leaves and their observed error rate. If a better split is found, the leaves of that split are deactivated, and their statistics are preserved. A new split is then performed, creating new leaves. If, during split re-evaluation, a previously deactivated split is found to be the best option, the saved statistics are restored rather than starting the process from scratch.

Overview. Table 1 summarizes how various algorithms implement different pre-pruning strategies, categorized by whether they adapt rules, grace periods, tie thresholds, or activity-based controls. As shown, DFDT offers the most comprehensive integration of these mechanisms, seamlessly combining them in a cohesive manner, as will be discussed.

Year	Model	Activity	Rules	$n_{\min}$	τ
2000	VFDT	X			
2005	VFDT-τ		X		X
2011	M-VFDT		X		X
2018	SVFDT		X		
2018	OSM			X	
2018	VFDT-$n_{\min}$			X	
2020	REG-VFDT		X		
2022	GAHT	X			
2026	DFDT	X	X	X	X

Table 1: Incremental decision trees

Methodology

DFDT modifications are here described in detail, while referring to the pseudo-code in Algorithm 1. The initialization phase creates the root node structure of the decision tree (Alg. 1, Step 2, 3), sets up estimators (Alg. 1, Step 4), and instance counters (Alg. 1, Step 5). This initialization involves only constant-time operations, resulting in a computational complexity of $O(1)$, independent of the number of instances or features. The main loop iterates over each instance (X, y) in the data stream S, running N times in total. For each instance, several steps are performed. First, the instance is routed through the tree to the corresponding leaf node, where the prediction $\hat{y}$ is obtained (Alg. 1, Step 7). Assuming a balanced tree, the depth of traversal is $O(\log_B |LH|)$, where $|LH|$ is the number of leaf nodes and B is the branching factor. The prediction at the leaf node is a constant-time operation, $O(1)$. Following this, the instance count and feature estimators are updated at the leaf node (Alg. 1, Steps 8-10). This update takes $O(F)$ time.

While the HB ensures that splits are made with statistical confidence, it treats all nodes equally in terms of expansion, despite the fact that not all nodes contribute equally to the decision-making process. To improve VFDT's efficiency in allocating computational resources, a notion of relative importance can be introduced to dynamically adjust the rate of expansion at each node. To assess the activity level at each node, a fraction parameter is calculated as:

$$\mathrm{f} = \frac{(n_l - n_{\text{leaf}_l}) \times |LH|}{n - n_{\text{tree}_l}} \tag{1}$$

where $(n_l - n_{\text{leaf}_l})$ is the number of instances observed at a particular leaf l since its creation, $n - n_{\text{tree}_l}$ the total instances observed by the tree since the creation of leaf l, and $|LH|$ the current total number of leaves. If the fraction is below a threshold $f_{\text{deactivate}}$, the leaf node is deactivated, halting further splits to save on resources (Alg. 1, Steps 12), which is a constant-time operation, $O(1)$. If the metric falls above a threshold f_{expand}, a boolean saves the intention to apply a more aggressive expansion strategy, accelerating the growth of important nodes (Alg. 1, Step 14).

To avoid costly evaluations, the algorithm only checks for potential splits once a leaf node accumulates $n_{\min}$ instances since the last splitting opportunity (Alg. 1, Step 16). Then a split attempt is performed (Alg. 1, Step 18), with its pseudocode detailed in Algorithm 2. This check involves computing $G(\cdot)$ values, requiring $O(F)$ time, and sorting the $G(\cdot)$ values (Alg. 2, Step 2), yielding a time complexity of $O(F \log F)$. Next, the algorithm verifies whether the HB or tie-breaking threshold is satisfied (Alg. 2, Step 4). While in the VFDT, the tie threshold (τ), which effectively controls minimum growth speed, remains static, DFDT, instead, dynamically sets the tie threshold as the mean of the Hoeffding Bound values computed over the most recent k instances (Yang and Fong 2011). This reduces the need for trial-and-error in selecting a fixed threshold, optimizing tree performance in dynamic environments. As the data stream becomes noisier, the adaptive τ stabilizes the model. If the HB is satisfied, DFDT utilizes adaptive expansion modes, deter-

Algorithm 1: Dynamic Fast Decision Tree (DFDT)

procedure DFDT(S: data stream, δ: confidence level, f_{expand}: threshold for skipping, $f_{\text{deactivate}}$: threshold to deactivate leaves)
  $DFDT \leftarrow$ root
  $LH \leftarrow$ hash of leaves
  Initialize $H_{LH_{\text{stat}}}, n_{\text{stat}}, H_{\text{stat}}, G_{\text{stat}}, HB_{\text{stat}}, n_{\min}$
  $n, n_{\text{l}}, n_{\text{check}_{\text{l}}}, n_{\text{leaf}_{\text{l}}}, n_{\text{tree}_{\text{l}}} \leftarrow 0$ where $l =$ root
  for each (X, y) in S **do**
    Route (X, y) to leaf l and obtain prediction $\hat{y}$
    Update l feature estimators and class distribution
    $n \leftarrow n + 1$
    $n_l \leftarrow n_l + 1$
    if $\frac{(n_l - n_{\text{leaf}_l}) \times |LH|}{n - n_{\text{tree}_l}} < f_{\text{deactivate}}$ **then**
      Deactivate l
    else if $> f_{\text{expand}}$ **then**
      GrowFast $\leftarrow$ True
    end if
    if (impure at l) and $(n_l - n_{\text{check}_l} > n_{\min})$ **then**
      Update HB_{stat} with ϵ
      if CANSPLIT(GrowFast) **then**
        Replace leaf l with a split node
        for each leaf branch b of the split **do**
          Initialize estimators
          Set post-split distribution at b
          Update LH
          $n_b, n_{\text{leaf}_b}, n_{\text{check}_b} \leftarrow \sum_{c \in \mathcal{C}} n_b^{(c)}$
          $n_{\text{tree}_b} \leftarrow n$
        end for
      else
        $n_{\text{check}_l} \leftarrow n_l$
        if $\overline{HB}_{\text{stat}} < \Delta G < \epsilon$ **then**
          $n_{\min} \leftarrow \left\lceil \frac{R^2 \ln(1/\delta)}{2(\Delta G)^2} \right\rceil$
        else if $\Delta G < \overline{HB}_{\text{stat}} < \epsilon$ **then**
          $n_{\min} \leftarrow \left\lceil \frac{R^2 \ln(1/\delta)}{2(\overline{HB}_{\text{stat}})^2} \right\rceil$
        end if
      end if
    end if
  end for
end procedure

Algorithm 2: Split conditions

procedure CANSPLIT(GrowFast = False),
  $G_{\text{best}} = \max(SortedG(\cdot))$
  $G_{\text{second best}} = \max(SortedG(\cdot) \setminus \{G_{\text{best}}\})$
  if $G_{\text{best}} - G_{\text{second best}} \geq \epsilon$ or $\epsilon < \overline{HB}_{\text{stat}}$ **then**
    if GrowFast **then**
      $C1 \leftarrow H_l \geq \overline{H}_{\text{stat}} + \sigma(H_{\text{stat}})$
      $C2 \leftarrow G_{\text{best}} \geq \overline{G}_{\text{stat}} + \sigma(G_{\text{stat}})$
      if C1 and C2 **then**
        return True
      end if
    end if
    $C3 \leftarrow H_l \geq \overline{H}_{LH_{\text{stat}}} - \sigma(H_{LH_{\text{stat}}})$
    $C4 \leftarrow H_l \geq \overline{H}_{\text{stat}} - \sigma(H_{\text{stat}})$
    $C5 \leftarrow G_{\text{best}} \geq \overline{G}_{\text{stat}} - \sigma(G_{\text{stat}})$
    $C6 \leftarrow n_l \geq \overline{n}_{\text{stat}}$
    Update H_{stat} with H_l
    Update G_{stat} with G_{best}
    Update n_{stat} with n_l
    if C3 and C4 and C5 and C6 **then**
      return True
    end if
  end if
  return False
end procedure

mined by the fraction-determined boolean. For nodes with moderate activity, DFDT applies four conservative splitting constraints (da Costa, de Leon Ferreira, and Duarte 2018), ensuring controlled growth (Alg. 2, Step 12-15). For highly active nodes, i.e., when the fraction value exceeds 2, DFDT also checks two skipping conditions (Alg. 2, Step 5-11).

The four primary constraints governing the split operate on the instance count, global entropy, historical entropy, and historical information gain. The instance count constraint ensures that splits only occur when the leaf has accumulated sufficient data, by checking if the current instance count n_l at the leaf l meets or exceeds the mean instance count recorded across all leaves under VFDT-satisfied conditions. Conversely, the remaining three constraints are evaluated using the same general formula $\varphi(x, X)$, which determines that the current metric value x at a leaf is significantly different from reference values X when bigger than the mean of the values in X minus a standard deviation:

$$\varphi(x, X) = \begin{cases} \text{True, if } x \geq \bar{X} - \sigma(X) \\ \text{False, otherwise} \end{cases} \tag{2}$$

The global entropy constraint checks if the current entropy H_l of a leaf l is high compared to the overall entropy of all leaves in the tree, denoted as $H_{LH_{\text{stat}}}$. Here, the set X consists of entropy values from all leaves, recorded continuously across the tree. The historical entropy constraint examines whether the current entropy H_l at a leaf l is significantly higher than historical entropy values recorded across all leaves at the times when VFDT conditions were met, denoted as H_{stat}. This ensures that splits are performed only if the leaf's entropy has increased meaningfully over time. The historical information gain constraint assesses whether the current information gain G_l of the best splitting feature at a leaf l is notably higher than historical information gain values recorded across all leaves when VFDT conditions were met, denoted as G_{stat}. This avoids low-value splits.

Together, these constraints allow DFDT to restrict tree growth in nodes of moderate activity to instances with statistically significant differences. For highly active nodes, two skipping conditions are calculated according to $\omega(x, X)$, which permits a split when the current metric x, i.e. the entropy H_l and information gain G_{best}, is at least one standard deviation above the mean historical values in X:

$$\omega(x, X) = \begin{cases} \text{True, if } x \geq \bar{X} + \sigma(X) \\ \text{False, otherwise} \end{cases} \quad (3)$$

These splitting conditions work as a skipping mechanism to the conservative constraints. In these cases, splits are allowed to proceed directly if the current entropy and information gain are significantly higher than historical means. Subsequently, the statistics for entropy, information gain, HB, and the number of instances seen are updated (Alg. 2, Step 16-18). All of these are constant-time operations. If the conditions are satisfied, a split is triggered (Alg. 2, Step 18). This split involves navigating the tree, sorting features, and creating new nodes, with the added complexity of reinitializing feature estimators for each branch (Alg. 1, Step 19-25). The time complexity becomes $O(F \log F + B \cdot F)$, where $B \cdot F$ represents the initialization cost for the new branches. If the split attempt fails at any point (Alg. 2, Step 23), the algorithm resets the instance count of the last checked split (Alg. 1, Step 28), and recalculates the grace period n_{min} based on specific conditions at each leaf (Alg. 1, Steps 29-32). This decision is based on the tie threshold τ, the difference in information gain between the top attributes ΔG and the statistical confidence bound ϵ, derived from the HB (García-Martín et al. 2018), with two possible scenarios. In the first scenario, ΔG is smaller than ϵ, but still exceeds τ, suggesting that more data is needed to confirm the observed gain. In this case, n_{min} is increased to allow additional data accumulation, ensuring that the gain difference becomes statistically significant in the subsequent evaluation. In the second scenario, ΔG is below τ, indicating that the attributes are statistically similar, but ϵ is still above τ. Here, n_{min} is further increased to delay the split until more data is gathered, allowing ϵ to shrink sufficiently. Thus:

$$n_{\text{min}} = \begin{cases} \left\lceil \frac{R^2 \ln(1/\delta)}{2(\Delta G)^2} \right\rceil, & \text{if } \tau < \Delta G < \epsilon \\ \left\lceil \frac{R^2 \ln(1/\delta)}{2\tau^2} \right\rceil, & \text{if } \Delta G < \tau < \epsilon \end{cases} \quad (4)$$

where δ is a user-defined confidence parameter, and R is the range of heuristic values. This adjustment minimizes unnecessary computations and improves energy efficiency, while maintaining the model's accuracy. Since split attempts (Alg. 1, Steps 16–35) occur only once every n_{min} instances, their overall contribution to the time complexity is weighted by the grace period factor. Thus, the total time complexity is: $N \cdot O(\log_B |LH| + F) + \frac{N}{n_{\text{min}}} \cdot O(F \log F + B \cdot F)$.

Experiments

All models and algorithm components were implemented within the pystream library[1], a Cython-based tool for data stream mining, reducing runtime overhead by compiling Python code into C. For evaluation, a diverse set of real-world datasets obtained from the USP Data Stream Repository[2] were used (Souza et al. 2020). These are shown in Table 2, encompassing both binary and multiclass classification tasks. The experiments were conducted using a prequential (test-then-train) evaluation strategy (Gama, Sebastiao, and Rodrigues 2009), monitoring the interrelationships among *accuracy*, *memory* usage, and *runtime*. All algorithms were optimized for each dataset using a grid search strategy, with grace periods (n_{min}) set to 100, 400, and 1000, and tie-splitting thresholds (τ) configured to 0.01, 0.05, and 0.1 as the hyperparameter settings. The Hoeffding bound confidence level was fixed at 1×10^{-7} across all experiments. DFDT's adaptive control of grace periods and tie thresholds removes the need for tuning these specific hyperparameters, as they govern tree growth and thus become implicitly determined by the HB confidence. However, DFDT introduces two fixed activity thresholds. While this raises questions about their adaptivity, it is important to note that, in contrast to n_{min} and τ, adapting these activity thresholds through the HB confidence is inappropriate, because activity operates conditionally on HB-driven splits and modulates leaf deactivation in ways that counteract tree growth. In practice, consistent with findings from GAHT (Garcia-Martin et al. 2022), we observed that a leaf-deactivation threshold below 2% of the expected instance count ($f_{\text{deactivate}} = 0.02$) and a skip-check threshold at twice the expected count ($f_{\text{expand}} = 2$) provide robust performance across datasets. Majority class prediction was used at the leaf level for all models.

Name	Classes	Features	Examples
OZONE	2	72	2,534
NOAA	2	8	18,159
METER	10	96	22,950
ELEC	2	8	45,312
RIALTO	10	27	82,250
POSTURE	11	3	164,860
KDDCUP99	23	41	494,021
COVER	7	54	581,012
POKER	10	11	829,201

Table 2: Experimental datasets

Since DFDT integrates multiple heuristics (Rules, Activity, τ, and n_{min}), it is essential to conduct an ablation study to assess the individual and combined contributions of these components to runtime, memory usage, and accuracy. This analysis helps identify Pareto-efficient variants of the algorithm. To ensure fair comparison across datasets of varying scales, all results were normalized on a per-dataset basis. For interpretability, normalized memory and runtime values were inverted, so that higher scores consistently reflect better performance across all metrics. Figure 2 presents coefficient estimates from linear regression models, each fitted separately for the three target outcomes. Interaction terms were included to capture the synergistic effects between heuristics. Individually, the adaptive grace period, activity-aware pruning, and stricter splitting rules did not produce statistically significant accuracy–memory–runtime trade-offs. Moreover, the combinations of Rules × Activity and Rules ×

[1]https://github.com/vturrisi/pystream

[2]https://sites.google.com/view/uspdsrepository

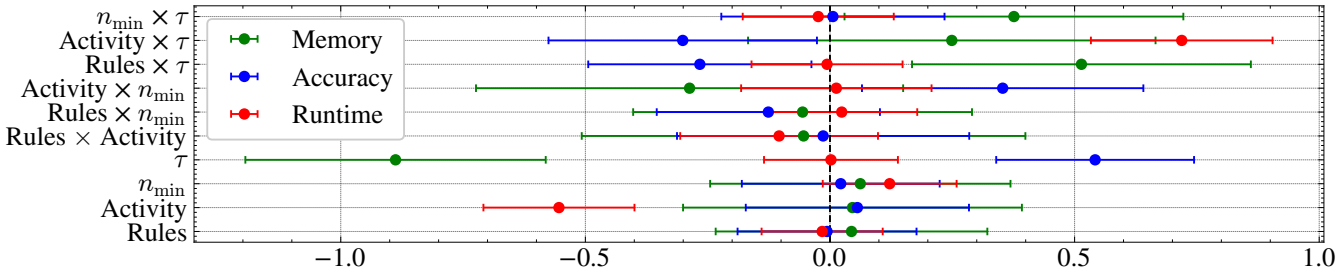

Figure 2: Main and interaction effects of DFDT components

$n_{\min}$ also demonstrated stable trade-offs, informing the design of DFDT_{Low} and $\text{DFDT}_{\text{Medium}}$, respectively. In contrast, the interaction between Activity × $n_{\min}$ and the inclusion of an adaptive τ significantly improved accuracy, albeit at the expense of memory efficiency. Additionally, τ's interactions with other heuristics exhibited inconsistent effects on accuracy, suggesting a more complex dynamic. As a result, $\text{DFDT}_{\text{High}}$ activates all heuristics to maximize accuracy, potentially at the cost of runtime or memory consumption.

Model	Rules	Activity	$n_{\min}$	τ
DFDT_{Low}	X	X		
$\text{DFDT}_{\text{Medium}}$	X		X	
$\text{DFDT}_{\text{High}}$	X	X	X	X

Table 3: Pareto-efficient DFDT variants

Figures 1 and 3 show the average performance across all datasets, plotting prequential accuracy against worst-case memory usage (MB) and per-instance computational time (µs), respectively. Table 4 reports the detailed results for each dataset. Among the baseline methods, VFDT exhibits moderate predictive performance while being the slowest and most memory-intensive model, underscoring its inefficiency across both metrics. Its modified variant, VFDT-$n_{\min}$, offers a modest reduction in memory consumption and a notable improvement in runtime, with minimal impact on accuracy. SVFDT demonstrates the lowest memory usage and maintains competitive runtime with VFDT-$n_{\min}$, however this efficiency is achieved at the expense of predictive accuracy. In contrast, the proposed DFDT variants delineate a clear Pareto frontier with respect to memory usage and share a competitive frontier with VFDT-$n_{\min}$ in terms of runtime. Specifically, DFDT_{Low} stands out as the most resource-efficient model, exhibiting lower runtime and memory consumption than SVFDT while maintaining comparable accuracy, making it particularly suitable for deployment in resource-constrained environments. The $\text{DFDT}_{\text{Medium}}$ variant achieves a favorable trade-off, significantly improving accuracy while preserving strong efficiency in both runtime and memory. Finally, $\text{DFDT}_{\text{High}}$ attains the highest overall accuracy, accompanied by a modest increase in computational time and a more pronounced increase in memory usage. Thus, only appropriate when resource constraints are less critical.

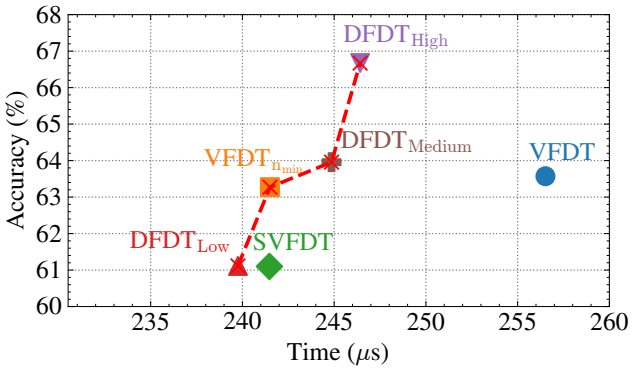

Figure 3: *Accuracy* x *Runtime*

To assess the statistical significance of the observed differences, we applied the Friedman test followed by the Nemenyi post-hoc test. The resulting Critical Difference Diagrams (CDDs), shown in Figures 4, 5, and 6, display the algorithm rankings at a 95% confidence level. In these diagrams, lower values correspond to better average ranking across datasets. The CDD for accuracy reveals that $\text{DFDT}_{\text{High}}$ achieves the best overall rank, and is the only method that significantly outperforms SVFDT. This result confirms the effectiveness of incorporating all heuristic components within DFDT to maximize predictive performance.

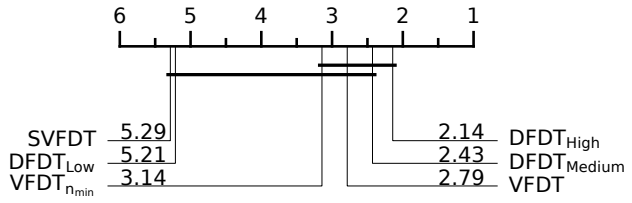

Figure 4: Nemenyi test - *Accuracy*

The ranking for memory consumption exhibits an inverse trend compared to accuracy. VFDT is the most memory-intensive algorithm, receiving the worst average rank. In contrast, DFDT_{Low} achieves the best overall rank. Notably, VFDT performs significantly worse than both DFDT_{Low} and

Method	NOAA	METER	ELEC	RIALTO	POSTURE	COVER	POKER	Avg. (Rank)
VFDT	73.1 ± 2.2 0.47 MB 43.2 µs	52.4 ± 2.3 14.19 MB 288.5 µs	**80.3 ± 1.5** 0.14 MB 49.8 µs	35.0 ± 4.9 3.37 MB 118.1 µs	50.5 ± 2.1 0.22 MB 78.5 µs	79.5 ± 2.5 0.80 MB 254.1 µs	74.2 ± 4.5 0.43 MB 963.4 µs	63.6 (2.79) 2.80 (5.36) 256.5 (5.57)
$VFDT_{n_{min}}$	72.6 ± 2.0 0.31 MB 35.0 µs	52.4 ± 2.1 10.37 MB 285.1 µs	79.4 ± 1.6 0.10 MB 38.4 µs	34.1 ± 4.3 2.87 MB **104.7 µs**	50.7 ± 1.5 0.21 MB 69.3 µs	78.3 ± 2.4 0.62 MB 241.2 µs	75.4 ± 4.5 0.43 MB 916.8 µs	63.3 (3.14) 2.13 (4.14) 241.5 (3.00)
SVFDT	71.2 ± 2.0 **0.15 MB** **33.0 µs**	52.3 ± 1.9 3.73 MB 283.0 µs	79.0 ± 0.6 **0.05 MB** **35.7 µs**	32.1 ± 3.4 0.87 MB 109.0 µs	48.8 ± 1.7 0.10 MB **69.0 µs**	74.6 ± 2.0 **0.39 MB** 255.3 µs	69.7 ± 2.0 0.38 MB **905.3 µs**	61.1 (5.29) 0.81 (1.93) 241.5 (2.21)
$DFDT_{Low}$	72.4 ± 1.8 0.16 MB 33.3 µs	52.3 ± 1.9 **2.76 MB** 279.1 µs	79.1 ± 0.9 0.07 MB 37.8 µs	33.9 ± 4.1 **0.54 MB** 110.2 µs	48.4 ± 1.1 **0.09 MB** 70.3 µs	72.7 ± 1.9 0.47 MB **239.8 µs**	69.0 ± 1.5 **0.37 MB** 907.8 µs	61.1 (5.21) **0.64 (1.71)** **239.8 (2.64)**
$DFDT_{Medium}$	**75.1 ± 0.3** 0.21 MB 35.3 µs	54.7 ± 0.5 2.92 MB **254.7 µs**	79.4 ± 0.6 0.10 MB 39.0 µs	**42.3 ± 0.5** 0.47 MB 112.3 µs	**52.3 ± 0.6** 0.20 MB 69.4 µs	70.9 ± 1.3 1.12 MB 260.5 µs	72.9 ± 5.4 1.23 MB 942.8 µs	63.9 (2.43) 0.89 (3.71) 246.4 (4.14)
$DFDT_{High}$	74.6 ± 0.5 0.21 MB 34.0 µs	**54.8 ± 0.4** 12.19 MB 280.5 µs	79.0 ± 0.4 0.05 MB 36.2 µs	41.7 ± 0.6 3.21 MB 105.7 µs	51.8 ± 1.0 0.20 MB 70.3 µs	**80.1 ± 1.1** 1.05 MB 255.3 µs	**84.7 ± 2.7** 1.23 MB 942.9 µs	**66.7 (2.14)** 2.59 (4.14) 244.9 (3.42)

Table 4: Prequential accuracy (%), memory usage (MB), and per-instance computational time (µs)

SVFDT, whereas $DFDT_{Medium}$ and $DFDT_{High}$ do not differ significantly, highlighting their well rounded performance.

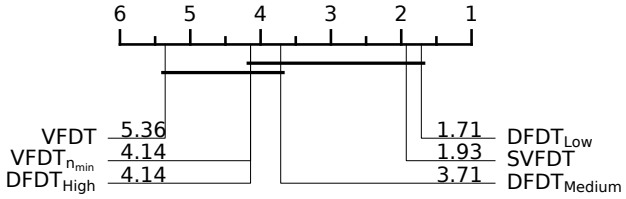

Figure 5: Nemenyi test - *Memory*

The CDD for runtime identifies VFDT as the slowest algorithm. Among the compared methods, only SVFDT demonstrated statistically significant better performance than VFDT. Although the variant $DFDT_{Low}$ exhibited a lower mean runtime than SVFDT, its ranking was not consistently superior across datasets.

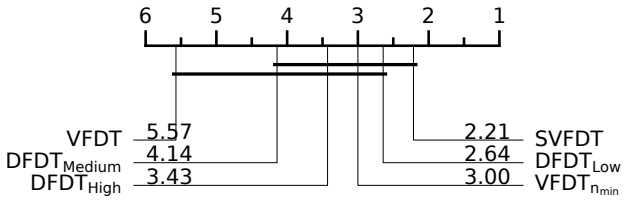

Figure 6: Nemenyi test - *Runtime*

Conclusions

Dynamic Fast Decision Tree (DFDT) combines different adaptive strategies from prior research to control decision tree growth. These features make DFDT particularly suitable for resource-constrained environments, such as IoT devices, edge computing applications, and real-time systems.

Extensive experimental evaluation, including ablation studies and statistical comparisons across a diverse set of data streams, demonstrated that DFDT variants consistently outperform or match baseline learners. In particular, $DFDT_{Low}$ was identified as the most resource-efficient configuration, achieving strong performance with minimal resource consumption. $DFDT_{Medium}$ offered a favorable trade-off between predictive accuracy and efficiency, while $DFDT_{High}$ achieved the highest accuracy overall, justifying increased memory and runtime costs when accuracy is the primary concern.

Unlike traditional VFDT-based learners, DFDT incorporates fine-grained control over growth and memory usage without sacrificing model adaptability. Moreover, its compatibility with existing ensemble frameworks allows it to serve as a drop-in replacement for VFDT, offering immediate benefits in large-scale, real-time stream mining applications. Future work may explore the integration of DFDT in different ensemble frameworks and investigate its performance on a broader range of datasets, including noisy imbalanced high-dimensional data streams.

Acknowledgments

Work funded by Portuguese Foundation for Science and Technology under Ph.D. scholarship PRT/BD/154713/2023 and project doi.org/10.54499/UIDP/00760/2020.

References

Barddal, J. P.; and Enembreck, F. 2020. Regularized and incremental decision trees for data streams. *Annals of Telecommunications*, 75: 493–503.

Bifet, A.; and Gavalda, R. 2009. Adaptive learning from evolving data streams. In *Advances in Intelligent Data Analysis VIII: 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31-September 2, 2009. Proceedings 8*, 249–260. Springer.

da Costa, V. G. T.; de Leon Ferreira, A. C. P.; and Duarte, J. M. 2018. Strict very fast decision tree: a memory conservative algorithm for data stream mining. In *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–7. IEEE.

Domingos, P.; and Hulten, G. 2000. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, 71–80. ACM.

Gaber, M. M.; Gama, J.; Krishnaswamy, S.; Gomes, J. B.; and Stahl, F. 2014. Data stream mining in ubiquitous environments: state-of-the-art and current directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(2): 116–138.

Gama, J.; Medas, P.; and Rodrigues, P. 2005. Learning decision trees from dynamic data streams. *Journal of Universal Computer Science*, 11(8): 1353–1366.

Gama, J.; Rodrigues, P. P.; Spinosa, E.; and Carvalho, A. 2010. Knowledge discovery from data streams. In *Web Intelligence and Security*, 125–138. IOS Press.

Gama, J.; Sebastiao, R.; and Rodrigues, P. P. 2009. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 329–338.

Garcia-Martin, E.; Bifet, A.; Lavesson, N.; König, R.; and Linusson, H. 2022. Green accelerated Hoeffding tree. *arXiv preprint arXiv:2205.03184*.

García-Martín, E.; Lavesson, N.; Grahn, H.; Casalicchio, E.; and Boeva, V. 2018. Hoeffding trees with nmin adaptation. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, 70–79. IEEE.

Gomes, H. M.; Barddal, J. P.; Enembreck, F.; and Bifet, A. 2017. A survey on ensemble learning for data stream classification. *ACM Computing Surveys*, 50(2): 1–36.

Holmes, G.; Richard, K.; and Pfahringer, B. 2005. Tie-breaking in Hoeffding trees. ECML/PKDD.

Hulten, G.; Spencer, L.; and Domingos, P. 2001. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 97–106.

Krawczyk, B.; Minku, L. L.; Gama, J.; Stefanowski, J.; and Woźniak, M. 2017. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37: 132–156.

Losing, V.; Wersing, H.; and Hammer, B. 2018. Enhancing very fast decision trees with local split-time predictions. In *2018 IEEE international conference on data mining (ICDM)*, 287–296. IEEE.

Lourenço, A.; Gama, J.; Xing, E. P.; and Marreiros, G. 2025a. In-context learning of evolving data streams with tabular foundational models. *arXiv preprint arXiv:2502.16840*.

Lourenço, A.; Rodrigo, J.; Gama, J.; and Marreiros, G. 2025b. On-device edge learning for IoT data streams: a survey. *arXiv preprint arXiv:2502.17788*.

Manapragada, C.; Webb, G. I.; and Salehi, M. 2018. Extremely fast decision tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1953–1962.

Neves, L.; Lourenço, A.; Cano, A.; and Marreiros, G. 2025. Online hierarchical partitioning of the output space in extreme multi-label data stream. *arXiv preprint arXiv:2507.20894*.

Núñez, M.; Fidalgo, R.; and Morales, R. 2007. Learning in Environments with Unknown Dynamics: Towards more Robust Concept Learners. *Journal of Machine Learning Research*, 8(11).

Souza, V. M. A.; Reis, D. M.; Maletzke, A. G.; and Batista, G. E. A. P. A. 2020. Challenges in Benchmarking Stream Learning Algorithms with Real-world Data. *Data Mining and Knowledge Discovery*, 34(6): 1805–1858.

Yang, H.; and Fong, S. 2011. Moderated VFDT in stream mining using adaptive tie threshold and incremental pruning. In *International Conference on Data Warehousing and Knowledge Discovery*, 471–483. Springer.