UCAT - U-TEST CERTAINTY ADAPTIVE TREE FOR REGRESSION ON NON-STATIONARY DATA STREAMS

Benedikt Stratmann
Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB)
Karlsruhe, Germany
benedikt.stratmann@iosb.fraunhofer.de

Jan Hermes
Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB)
Karlsruhe, Germany

Dr. Constanze Hasterok
Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB)
Karlsruhe, Germany

ABSTRACT

Cyber-physical systems (CPS) often rely on learned surrogate models whose performance degrades under non-stationary conditions. While many adaptive classifiers exist, adaptive methods for tabular regression are scarce. We propose UCAT, an adaptive model tree for online regression on non-stationary data streams in CPS. UCAT replaces the Hoeffding bound used by existing adaptive trees with a one-sided Wilcoxon–Mann–Whitney U-test on absolute prediction errors to select splits, and combines residual model trees with local linear models, Page-Hinkley drift detection, rival branches for subtree replacement, and exposure-based pruning. An optional twig-level threshold adaptation further refines split thresholds. On the SiD2Re benchmark (15 datasets, 7.680 stream variants), we compare UCAT and a twig-adapting variant (UCAT$_{twig}$) with FIMT-DD using prequential evaluation. UCAT achieves lower cumulative absolute error on 68% of streams, with an average reduction of 13.33%, and is preferred by normalized error metrics on most datasets, whereas FIMT-DD performs best mainly in stationary or high-noise regimes. These gains come at increased computational cost: UCAT is about 43% slower than FIMT-DD. Overall, UCAT provides an interpretable, streaming-capable regression method with statistically grounded split selection and targeted adaptation for CPS, suitable when improved accuracy and drift handling justify higher runtime.

Keywords Non-stationary data streams · Online regression · Adaptive model trees · Concept drift detection · Wilcoxon–Mann–Whitney U-test

1 Introduction

This work studies ever-evolving industrial cyber-physical systems (CPSs), with a focus on manufacturing settings such as “Industrie 4.0”, “Smart Manufacturing”, and “Smart Factory”. CPSs couple physical processes with networked computation and sensing, yielding heterogeneous, high-dimensional data streams that challenge conventional Machine Learning (ML) methods [9, 34, 21]. Among the ML tasks encountered in such systems, regression is most prevalent but less well studied than classification in this domain [21, 30]. At the same time, CPSs are data-rich yet knowledge-sparse, making interpretable models particularly attractive for monitoring and understanding system behaviour [9, 34].

This work introduces the U-Test Certainty Adaptive Tree (UCAT), an innovative incremental and **adaptive** tree learner that provides **interpretable** results in the context of **regression**. To achieve this, UCAT reuses established techniques of adaptive tree learners, develops a new method for the incremental growth of decision trees, and proposes a reformulation of known tree pruning and split decision adaption approaches. The resulting UCAT tree can grow based on statistical evidence, detect drifted subtrees, restructure these subtrees through reforming or substitution, and keep its internal structure consistent with current data samples to provide anytime interpretability.

2 Theoretical Background

Most ML methods assume a stationary context, i.e. where distributions and relations of system variables do not change over time. Decision tree learning methods [5, 28] and their incremental extensions [31, 11] also make this assumption. Complex physical systems [20] and those impacted by human actions [12] are not-stationary, leading to a decline in model performance when there are changes in the underlying distributions between the training and operational settings.

Problem Statement Non-stationary environments are characterized by their time changing data streams. These changes can often be expressed as distribution shifts in the data stream. Such so-called drifts refer to changes in the data distribution that can vary in the direction of their shift within the high-dimensional feature/target space, the speed at which they unfold, and the underlying cause driving them [2]. While there are different types of drifts they all have something in common. The data generating process $\mathcal{P}$ that provides the stream of data tuples $d_i = (x_i, y_i, t_i) \in X^n \times Y^m \times \mathbb{R}$ changes over time. $X^n \times Y^m$ describes the data space composed of process data/features X and target values/labels Y. The data generating process itself can be viewed as a sampling from a distribution $p_t(x, y)$ at a specific time t. If $\mathcal{P}$ is part of a non-stationary environment we can *not* assume that $\forall t_1, t_2 : p_{t_1}(x_1, y_1) = p_{t_2}(x_1, y_1)$ as unforeseen drifts may change the data generating distributions over time [10]. This distribution is also called the system concept [32].

The task of learning in a non-stationary environment (NSE) is focused on maintaining model accuracy over time while detecting, analyzing, or compensating for drifts that occur at different points or intervals of time. This work focuses especially on regression tasks most frequently encountered in CPS [21]. This results in $X \in \mathbb{R}$ and $Y \in \mathbb{R}$. The data stream D is therefore $D = \{d_i = (x_i, y_i, t_i) | i \in \mathbb{N}\} \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}$. The specifics on the learning and validation setup are further explained in Section 5 and Figure 2.

Methods deployed to an NSE need to be able to adapt to the drifts that occur. This adaption can be active or passive. Active adaption needs a change detection that acts as a trigger to start any kind of adaption in the model. Passive adaption assumes that the environment is constantly drifting and therefore is continuously in the process of adapting. These approaches represent the current state of the art for handling NSE within the ML context [30].

Objective: Find an interpretable model for the observed system behavior that can keep performance high under the influence of drifts and provides insights into the location of the drifts.

One solution to this problem specification are adaptive trees. Adaptive trees can be characterized by answering a few questions regarding their methodology: **How**, **where**, and **when** to **grow**? **What** to **prune**? **Where** and **how** to **adapt**? This section will present the general concepts used to answer these questions. Figure 1 shows how UCAT answers these questions on a high level view, while later sections will provide a detailed notation of the UCAT method.

How to grow? One fundamental concept of growing trees on data streams is the continuous transformation of leaves into split nodes with new associated leaves. Whenever a split is established, a leaf $\mathcal{L}$ is transformed into a split node $\mathcal{S}$ and two resulting new leaves $\mathcal{L}_{\leq}$ and $\mathcal{L}_{>}$. We use binary split operations throughout this work. A split node $\mathcal{S}$ always holds information about its child nodes and the decision it represents. A decision is composed of a split threshold value, which divides data into two disjoint feature subspaces, based on a feature dimension to apply the threshold value on. A leaf $\mathcal{L}$ on the other hand, contains information about the data that falls within the feature subspace it represents. Every node $\mathcal{N}$ of a tree is either a split node or a leaf.

For inference, the tree's structure has to be traversed until a leaf is reached. One can either use the model stored in the leaf for the prediction or use the Residual Propagation (RP) method. Using RP, each node of the tree that is traversed contributes to the prediction. For tree growth, the error residuals of the target dimensions substitute the original target values when the sample vector is passed to the child node. Meaning, a split node does not propagate the data tuple (x, y) consisting of a feature vector x and a target vector

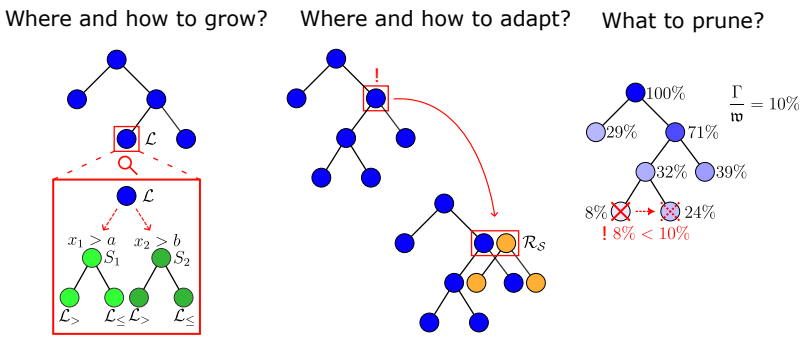

Figure 1: High level view on how UCAT answers some of the characterizing questions for adaptive tree learners. We consider binary splits and compare different split candidates for tree growth (left). We detect a drifted or invalidated subtree and propose a set of rivals for replacement (center). We evaluate data coverage of tree nodes and execute pruning (right).

y, to the corresponding child node. Instead the split node $\mathcal{S}$ uses its internal model $\mathcal{M}$ and propagates $(x, y - \text{Pred}_\mathcal{S}^\mathcal{M}(x))$. $\text{Pred}_\mathcal{S}^\mathcal{M}(x)$ is the prediction which is proposed by model $\mathcal{M}$ at $\mathcal{S}$ for input x. Hence, the children of a node serve to correct the prediction errors of the parent node.

Where to grow? Whenever a leaf is transformed into a split node, a split heuristic $\mathcal{G}$ is used to select the split threshold and split dimension. A set of possible splits is sampled and the split with the best value of $\mathcal{G}$ is chosen. $\mathcal{G}$ usually measures how similar samples are within each of the resulting leaves or how dissimilar they are between the leaves. Depending on which heuristic is chosen, it is either minimized or maximized. Various split heuristics are presented and reviewed in [5, 6, 27, 7] and methods for split sampling can be found in [8].

For this work, Gaussian split sampling [8] and the variance reduction split heuristic [5] are used for all tree learners. Gaussian sampling has a high probability to find a split that evenly distributes the available samples onto the leaves, while the split heuristic aims to increase target value homogeneity independently of any assumptions. These approaches are chosen because of their tendency to produce stable decisions when used in NSE. Other sampling approaches either introduce assumptions about the underlying data distributions or result in splits, which are unbalanced and therefore have to be removed shortly after their creation.

When to grow? How much information is needed at a leaf to transform it into a split node? A common approach is to use user-defined thresholds. These may be the minimal number of samples needed in a leaf, for it to be transformed, or a requirement on the number of samples that need to reside in the new leaves generated after the transformation.

As this approach requires extensive domain knowledge, it is not always applicable. The next most common approach is to use statistical properties found within the data to trigger a split decision. A prevalent decision criterion is the Hoeffding's bound as introduced in [11]. Depending on the current amount of samples and the dissimilarities expressed through the split heuristic, this approach infers a certainty that a proposed split is suited or even close to optimal. Rutkowski et al. [29] identify **shortcomings** of Hoeffding's Bound for split selection. These shortcomings are related to the assumptions of independence between the compared data sets, which are violated in practice for both categorical and numerical target values. While Rutkowski et al. propose a new approach for the classification domain, regression remains without a solution until now. We propose a new split selection method that can be used in regression tasks. This split selection approach uses the **Wilcoxon–Mann–Whitney U-test [33, 24]**. The U-test is a rank-based, non-parametric test that compares two sample populations X and Y by counting the number of pairwise order violations, i.e., whether a randomly chosen element from X tends to be larger or smaller than one from Y. Under mild regularity conditions the test statistic is approximately normal, which enables straightforward computation of one-sided p-values. We use the U-test to determine, with user-defined confidence δ, whether the absolute-error distribution induced by the best candidate split is stochastically smaller than that of the runner-up candidate. A full formalization is provided in Appendix A. For how the U-Test is applied in split selection see subsection 4.1.

Where to adapt? If adaption would be allowed "everywhere" all the time, it would result in passive adaptive behavior. However, for the sake of computational costs and/or data conservation active adaption should be used. The used change detectors provide information on the parts of the tree that are changing or have

become invalid. The change detector of choice for this work is the Page-Hinckley Test (PHT) [25]. The PHT was developed as a method to measure changes in the behavior of continuous processes. In this work, we utilize PHT as a change detection method based on model errors. The used formulation of the test is taken from [17]. A version that is consistent with the notation used in this work can be found in Section B.

To answer the remaining questions that characterize adaptive trees, one has to take a deeper look into individual methods, as pruning and adaption are usually dependent on tree structure and how growth and inference are handled.

3 Related Work

The domain of adaptive regression trees is fairly sparse, consisting of Fast and Incremental Model Tree with Drift Detection (FIMT-DD) [18] and associated methods [16, 19, 17], while the domain of classification presents a variety of methods. The distinguishing factor, that makes the use of a classification tree in regression tasks non trivial, is the measurement of when a split improves the trees capabilities. This is such an embedded part of tree learners, that changing it to comply with regression requires many additional changes in the tree methodology.

Very Fast Decision Tree learner (VFDT) [11] and Hoeffding Option Tree learner (HOT) [26] are well-known and established incremental classification tree learners. VFDT uses the Hoeffding's bound for the decision when a leaf should be transformed to a split node. HOT uses the same bound while exploring multiple splits at one node at the same time. Both methods only aim at continuously growing a decision tree without a strategy for handling drifts. Current approaches for growing and adapting trees in data streams take into account adaptation through various means, from reconstructing tree branches to reforming existing branches. Expanding upon the foundational principles of VFDT [11] and HOT [26], we view a library of methods. This includes methods such as Very Fast Decision Tree learner for continous data (VFDTc) [13], Concept-adapting Very Fast Decision Tree learner (CVFDT) [15], Ambiguous Concept-adapting Very Fast Decision Tree learner (aCVFDT) [22], Hoeffding Adaptive Tree learner (HAT)-Adaptive Windowing (ADWIN) [3] and Adaptive Hoeffding Option Tree learner (AHOT) [4]. While all of these methods are based on the Hoeffding's Bound for deciding on splits their approaches display differences in three distinct fields: detecting and adapting drifted parts of the tree as well as pruning superfluous parts of the tree. As mentioned previously Rutkowski et al. [29] show that the Hoeffding's Bound is **not suited** for split selection, which prompted the search for an alternative approach and the development of UCAT.

In this work, FIMT-DD [18] serves as a point of reference, since its structure and methodology are comparable to the UCAT method we propose and both are designed to handle regression tasks. Both methods are active adapters that are implemented to use the PHT as the change detector. Additionally, both methods aim to substitute the original part of the tree with a new branch that is grown simultaneously. Lastly, both methods have model tree capabilities, meaning they are intended to be used with exchangeable knowledge representations for the subspaces represented by their leaves. Popular choices for such knowledge representations are linear or kernel regressions as well as small multi-layer perceptrons.

4 U-Test Certainty Adaptive Tree

On a high level view, U-Test Certainty Adaptive Tree (UCAT) maintains a rooted, ordered tree $\mathbb{T} = \{\mathcal{N}\}_\ell$ whose nodes $\mathcal{N}$ are either split nodes $\mathcal{S}$ or leaves $\mathcal{L}$. Split nodes store pointers to exactly two children, their parent as well an an optional pointer to a rival node. Leaves only store pointers to their parent. Learning consists of updating local models and statistics along this path and, at scheduled intervals, performing three operations:

- *growth*, where leaves are turned into splits using U-test-based significance tests
- *adaptation*, where drifted subtrees spawn rival branches
- *pruning*, where low-exposure nodes are removed via a binomial test.

The UCAT aims to incorporate the advances and desirable characteristics of adaptive classification trees with a well defined approach to determine if a new split is to be established. Lets quickly answer the characterizing questions for which UCAT uses established paradigms.

How to grow? Our tree learner is based on RP with binary splits and model tree functionalities, where each node uses a local linear regression model. The data stream is viewed through a window $\mathcal{W}$ of currently

relevant data that is used to infer and validate the current state of our tree. $\mathcal{W}$ only holds the most recent $|\mathcal{W}|$ samples, where $|\mathcal{W}|$ is defined by the user. Generally, every data point in $\mathcal{W}$ is sorted into one leaf of the tree to be used in the prediction process. For each split node, the data that was used to establish the split is stored, independent of whether this data is currently within the data window $\mathcal{W}$.

Where to grow? Gaussian split sampling [8] and the variance reduction split heuristic [5] are used in this work. This configuration proves to be robust across different learner types for the NSE we evaluate for. Other split heuristics like SMOTI [6], IMTI [27] and SUPPORT [7] increase runtime complexity while providing no or negative effect on model performance in preliminary tests. The same results are observed when using small neural networks or Gaussian processes instead of linear regression in the leaves to infer predictions.

Where to adapt? The change detection method selected for this work is the PHT [25]. Selecting a different or more recent change detector [23], like EDDM [1], may prove advantageous, but is beyond the scope of this work. Given that split nodes were introduced at a certain point in the past, our interest lies in contrasting the characteristics of the data upon which the split was originally determined with the data currently residing in $\mathcal{W}$. When using the PHT, we view the data the split was decided on and the data in $\mathcal{W}$ as a sequential time series.

4.1 When to grow?

Instead of using the Hoeffding's bound for deciding on splits, like FIMT-DD and the widely cited CVFDT, UCAT employs the U-Test to decide if a leaf should be transformed into a split node. For our proposed method, we compare the error populations $\mathcal{E}_{\mathcal{S}_1}$ and $\mathcal{E}_{\mathcal{S}_2}$ produced by the best split $\mathcal{S}_1$ and second best split $\mathcal{S}_2$ according to a preapplied split heuristic $\mathcal{G}$. This approach is based on the assumption that a split should improve the tree's absolute prediction error. The error population $\mathcal{E}_{\mathcal{S}}(\mathcal{W})$ of a split node $\mathcal{S}$ on a window of data $\mathcal{W}$, that depends on the tree learner method, is defined by

$$\mathcal{E}_{\mathcal{S}}(\mathcal{W}) = \{\|y - \mathrm{Pred}_{\mathcal{S}}(x)\|_1 \,|(x, y) \in \mathcal{W}\}, \tag{1}$$

where (x, y) is a data point of features x and targets y of the data window $\mathcal{W}$ and $\mathrm{Pred}_{\mathcal{S}}(x)$ is the target prediction given by a specific split node $\mathcal{S}$. To determine whether $\mathcal{E}_{\mathcal{S}_1}(\mathcal{W})$ compared to $\mathcal{E}_{\mathcal{S}_2}(\mathcal{W})$ indicates smaller prediction errors for $\mathcal{S}_1$, the one-sided version of the U-Test is used. As the U-Test can be slow to compute for large amounts of samples, we limit the number of computations by using a split heuristic $\mathcal{G}$ as a filter to only compare the most promising splits. The best split candidates for each feature dimension are the split candidates with the smallest $\mathcal{G}$ value. From them, only the best two are compared using the U-Test.

Rationale for using the U-Test. Alternative significance tests for split selection include parametric tests (e.g., t-tests) and tests on split heuristics (e.g., Hoeffding bounds on variance reduction). We choose the one-sided Wilcoxon–Mann–Whitney U-test because it is

- non-parametric and does not assume a specific residual distribution,
- robust to outliers and heteroskedastic errors, which frequently occur in CPS data streams, and
- directly compares the absolute-error distributions of competing splits rather than relying on a surrogate criterion.

Hoeffding-based criteria implicitly assume independent, bounded random variables and are applied to quantities (such as impurity or variance) that are not independent across candidate splits [29], which can lead to overconfident split decisions as real world scenarios frequently violate these assumptions. In contrast, the U-test provides a conceptually simple, distribution-free way to express certainty that one split produces systematically smaller prediction errors than another.

4.2 How to adapt?

Other methods like for example FIMT-DD and CVFDT handle adaption after a change detection by using alternate trees. UCAT applies a similar approach to alternate trees, the so-called rivals. Alternates and rivals are handled like temporary options. If a change is detected at a split node $\mathcal{S}$ we introduce a rival node next to the original split node. The rival branch is grown with all data in $\mathcal{W}$ that is assigned to $\mathcal{S}$, but not with the data upon which the split decision in $\mathcal{S}$ was determined. Note that a rival may also seed a rival itself at some point.

Besides the similarities between alternates and rivals, their goals differ. Alternate subtrees are typically grown passively and are intended to fully replace the original subtree once enough evidence accumulates.

In contrast, rivals are explicitly treated as competing hypotheses that share a limited data budget: only the best-performing rival receives new samples, which constrains memory growth and encourages rapid elimination of inferior alternatives. As soon as rival and original branch are associated with the same number of samples, the original branch and the rival engage in competition. When a new labeled sample is introduced to the tree for learning, UCAT always assigns it to exactly one leaf. Associating samples to exactly one leaf differs from what can be seen with option nodes in other works focused on adaptive tree learners [26, 19]. A new sample is only propagated to the rival that can infer the most accurate prediction w.r.t. a predefined metric, in our work this is the sum over the absolute error for all target dimensions. This approach provides more data to the most accurate of the competing rivals, starving the rest of them over time, which are then pruned by following the later described exposure based pruning method. The original split node $\mathcal{S}^*$ along with the rival and rival's rival are the set of all competitors $\mathcal{R}_{\mathcal{S}^*}$ that try to make the best prediction to get new samples assigned to themselves. To prevent the unwanted passive adaption of the tree through the used RP, the subtree rooted at $\mathcal{S}^*$ is prohibited from transforming leaves to split nodes as long as a rival exists. When using the tree for inference, the contribution of the original subtree and the rival subtree are weighted. The weight of a rival $\nabla \in \mathcal{R}_{\mathcal{S}^*}$ is defined by:

$$w(\nabla) = {}^{\hat{w}(\nabla)}/_{\sum_{\nabla' \in \mathcal{R}_{\mathcal{S}^*}} \hat{w}(\nabla')} \quad \text{with} \quad \hat{w}(\mathcal{N}) \quad = \frac{1}{\frac{1}{|\mathcal{W}|}\left(\sum_{(x,y)\in\mathcal{W}} |\text{Pred}_{\mathcal{N}}(x) - y|\right)^{(2\cdot(|\mathcal{R}_{\mathcal{S}^*}|-1))}}$$

Rivals in the tree data structure. On the implementation level, each split node $\mathcal{S}$ maintains at most one pointer to its primary rival root $\mathcal{N}_r$. Each rival root $\mathcal{N}_r$ in turn maintains its own subtree structure (children, local models, statistics) and a back-pointer to the original node $\mathcal{S}^*$ that triggered its creation. Rivals are thus stored as parallel subtrees attached to the same parent, rather than as special leaf types.

Another new mechanism of UCAT is the **adaption of the split threshold**. We acknowledge that the methods by which the split candidates are proposed and scored are only approximations and may not have found the overall best split threshold value. These approximations are necessary, as the use of multi-dimensional numerical feature data allows for an infinite number of split candidates. Whenever UCAT tries to adapt its structure to the current $\mathcal{W}$ the split thresholds of several splits are reevaluated under the scope of the information provided by new data. This reevaluation is executed on split nodes that only hold leaves as their children. We call them "twigs" in the following. Testing twigs refers to the evaluation of possible changes in the split threshold and not the split dimension. We evaluate the tree's predictive performance for various scenarios of thresholds that are shifted up or down by Δ defined by

$$\Delta = \begin{cases} 0 & \text{if } |D_{\mathcal{T}}| > a \\ (R \cdot \hat{r}) \cdot (1 - \frac{|D_{\mathcal{T}}|+1}{a}) & \text{else,} \end{cases} \tag{2}$$

where R represents the difference between the smallest and the largest feasible split value for the selected split sampling method and $\hat{r}$ is a user defined value between 0 and 1 which limits the maximum step size dependent on R. $|D_{\mathcal{T}}|$ is the number of samples that traversed the twig $\mathcal{T}$. Lastly, a is a user-defined number of samples after which split adaption should be stopped. The primary purpose of a is to prevent reoccurring computations but it also directly influences Δ, leading to smaller steps with time. If any scenario for changing the threshold by Δ or $-\Delta$ proves more performant than the original w.r.t. the tree's absolute prediction error, the tree is restructured to comply with the new threshold. As new data from the data stream propagates through a twig, $|D_{\mathcal{T}}|$ increases resulting in smaller values for Δ which eventually becomes zero. The twig becomes more rigid, limiting its ability to make adjustments. We evaluate two types of UCAT, namely UCAT and $\text{UCAT}_{\text{twig}}$. The presented twig threshold adaption approach introduces additional runtime overhead and is evaluated as an additional functionality in $\text{UCAT}_{\text{twig}}$, while UCAT does not use this approach.

4.3 What to prune?

Pruning is based on the amount of data assigned to a part of the tree. To remain in the tree, each node needs a predetermined number of samples Γ, called exposure, to support its existence. This can be seen under the intuition of **usefulness**: If a node does not provide enough value to the tree and is not supported by data, the node should be pruned.

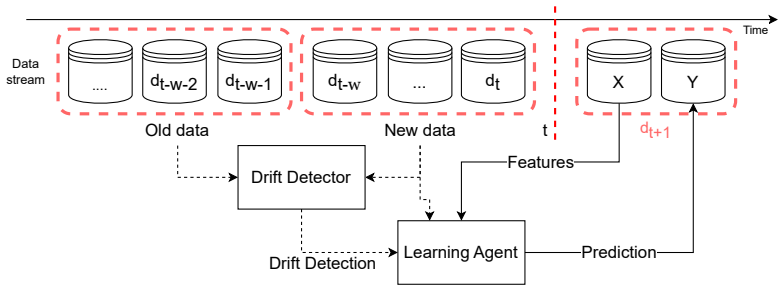

Figure 2: Training loop: Simulating the deployment of an adaptive learning agent on a continuous data stream.

Algorithm 1 UCAT Pruning($\mathcal{W}$, $\mathcal{N}$, δ)

```
// First, recursively prune all descendants
if N is NOT a leaf then
  for both children C of N do
    C ← UCATPruning(W, C, δ)
if N has a rival N_r then
  N_r ← UCATPruning(W, N_r, δ)
// Second, decide whether this node itself should survive
if N is a leaf then
  if N is useful then
    return N
  else
    return ∅
else
  for both children C of N do
    if C is ∅ then
      if N has a rival N_r then
        return N_r
      else
        Revert N to a leaf L with no descending nodes
        return L
```

We define the property of **usefulness** for a leaf $\mathcal{L}$ by the certainty threshold that a number of data points in $\mathcal{W}$ fall into $\mathcal{L}$. We test for this using a binomial test, where data points that fall into the leaf are seen as a success and the total number of trials is $|\mathcal{W}|$. If the hypothesis holds that the probability of success exceeds $\frac{\Gamma}{|\mathcal{W}|}$ we mark $\mathcal{L}$ as **useful**. If we can discard this hypothesis with a certainty of δ, we can conclude, that $\mathcal{L}$ does not exhibit enough samples over any other permutation of data in $\mathcal{W}$ with certainty δ. The pruning procedure is executed as seen in algorithm 1, which relates to the following intentions: The tree is pruned from the leaves to the root, the pruning is therefore implemented recursively. After applying the pruning rules on the leaves, the split nodes are checked to see if they hold dried off parts of the tree.

A part of a tree is dried off when it holds a pruned or nonexistent node. Any split node that dried off is returned to its leaf form and any samples in $\mathcal{W}$ previously assigned to its children are reassigned to the node. However, if a rival exists for a dried off split node, the rival takes the position of the dried off branch, discarding the obsolete node.

5 Experiments

We limit the analysis to the use of synthetic data to enable side-by-side comparisons of different NSE. The variety of established regression benchmarks specific to CPS is currently lacking [20, 23]. Consequently, this work leverages the new framework SiD^{2}Re, which is designed to simulate CPS-oriented synthetic data streams [14]. As of this time, one SiD^{2}Re benchmark *v1.0_230623* is published under `https://github.com/FraunhoferIOSB/SiD2Re`. This benchmark contains 15 datasets with about 11.000 different variations of NSE in total. A dataset refers to a specific initial concept that changes over time. Each variation contains a data stream with different evolution of the initial concept, different data distributions and a variety of different induced data and concept drifts. The number of variations that provide a specific property, like number of drifts or noise level, are listed in the appendix under Table 3. We exclude variations with more than 1000 data points from evaluation as the runtime of the methods is beyond the scope of this work. Larger datasets do not provide additional information on the evaluated performance metrics. This results in a total of 7680 variations in this work.

Each data stream is incrementally learned by each of the investigated tree learners. The prediction for the data point at time $t + 1$ is based on all data available until and including time t. Figure 2 shows a scheme of

the training loop. The drift detector has access to all previously recorded data and the learning agent acts upon the information of drift detection, new data and the feature information of the data instance that is to be predicted.

Given the number of methods, datasets and hyperparameters, exhaustive per-stream tuning is infeasible. We therefore perform a small grid search over a shared set of hyperparameters (frequency of node expansion checks, frequency of adaptations, decision certainty δ, and window size $|\mathcal{W}|$). For each method, we select the configuration that minimizes cumulative Absolute Error (AE) between predictions and ground truth. The chosen values and further experimental details are summarized in Section E.

6 Results

For the analysis of the multidimensional SiD^{2}Re data streams we use the CAE (Cumulative Absolute Error) to asses performance over the whole duration of the data stream and the RMSE (Root Mean Squared Error) for measuring the general prediction performance. The metrics are defined as

$$\text{RMSE}_{\mathbb{T}}(D) = \sqrt{\frac{1}{n} \sum_{(x,y,\cdot)\in D} (y - m(x))^2}$$

$$\text{CAE}_{\mathbb{T}}(D) = \sum_{(x,y,\cdot)\in D} |y - m(x)|$$

for the dataset D and target prediction $\mathbb{T}(x)$ of the completed tree model $\mathbb{T}$ that was learned and evaluated using D, as shown in Figure 2.

Each variation of a dataset presents an environment with different feature distributions and varying numbers of induced drifts, resulting in less and more complex learning tasks. Hence, just viewing the RMSE or CAE averaged over the variations of the dataset would provide a skewed interpretation. To compare the performance of different methods we inspect the mean of the *relative* error metric values. For each of the 7680 variations the metric value of the best performing method is used to normalize the remaining values:

$$\text{MRRMSE}_{\mathbb{T}} = \frac{1}{|V|} \sum_{D\in V} \frac{\text{RMSE}_{\mathbb{T}}(D)}{\min\limits_{\mathbb{T}^*\in M} \text{RMSE}_{\mathbb{T}^*}(D)}$$

$$\text{MRCAE}_{\mathbb{T}} = \frac{1}{|V|} \sum_{D\in V} \frac{\text{CAE}_{\mathbb{T}}(D)}{\min\limits_{\mathbb{T}^*\in M} \text{CAE}_{\mathbb{T}^*}(D)}$$

where V is the set of all evaluated variations and M is the set of evaluated tree learner methods (FIMT-DD, UCAT, UCAT$_{\text{twig}}$). MRRMSE (Mean Relative Root Mean Squared Error) and MRCAE (Mean Relative Cumulative Absolute Error) therefore have a smallest value of 1. For comparability with other methods beyond this work, the MCAE (Mean Cumulative Absolute Error) over all variations of each dataset is also provided. An overview of the results is given in Table 1. Boldface indicates the on-average best-performing method for each dataset.

Even though UCAT and UCAT$_{\text{twig}}$ outperform FIMT-DD in most experiments, the MCAE of FIMT-DD is smaller for two datasets. Figure 3 depicts the results for dataset c_13_multi_dim_splitter_tree_4_2 and shows that for the majority of the variations, FIMT-DD achieves better but comparable results to UCAT and UCAT$_{\text{twig}}$. However, there also exists a set of variations for which FIMT-DD performs significantly worse if compared to the results of UCAT and UCAT$_{\text{twig}}$ on the same variation. This disparity between experiment introspection and values of MCAE can be attributed to the fact that the MCAE gives more weight to experiments with overall higher error values. If we correct for this effect by normalizing by the metric of the best-performing method as done in MRRMSE and MRCAE, we see that the best results fall to UCAT and UCAT$_{\text{twig}}$.

UCAT achieves a smaller CAE in 68% of the experiments. Using UCAT over FIMT-DD for a variation improves the CAE on average by ~13.33%. The twig threshold adaption can frequently achieve better performance but can also achieve worse performance, while increasing runtime complexity. UCAT (UCAT$_{\text{twig}}$) takes about 43% (193%) more time to run an experiment than FIMT-DD. For more in depth runtime and memory analysis see the appendix section C. As the twig adaption is aimed at mitigating errors that the split heuristic could introduce, this leads to the hypothesis, that the used split heuristic is already sufficiently accurate, even in NSE. Proving this hypothesis is beyond the scope of this work.

Table 1: Overview of the average performance metrics for the viewed methods on the different datasets of the benchmark.

	MRCAE			MRRMSE			MCAE		
dataset	FIMT-DD	UCAT	$UCAT_{twig}$	FIMT-DD	UCAT	$UCAT_{twig}$	FIMT-DD	UCAT	$UCAT_{twig}$
c_10_multi_dim_tree_4_2	1.099	1.098	**1.094**	1.077	**1.039**	1.043	**1.97E+05**	2.02E+05	2.03E+05
c_10_steps_2_1	1.193	1.025	**1.018**	1.128	1.009	**1.008**	3.76E+05	3.11E+05	**3.10E+05**
c_13_multi_dim_splitter_tree_4_2	1.134	1.111	**1.088**	1.052	1.050	**1.045**	**2.27E+05**	2.43E+05	2.38E+05
c_13_sideways_helix_2_1	1.195	1.052	**1.047**	1.139	1.051	**1.021**	7.66E+05	6.63E+05	**6.59E+05**
c_15_multi_dim_fold_4_2	1.245	**1.015**	1.034	1.225	**1.006**	1.023	2.44E+06	**2.02E+06**	2.07E+06
c_20_double_fold_2_1	1.393	1.049	**1.032**	1.282	1.034	**1.026**	5.86E+05	4.53E+05	**4.48E+05**
c_26_layered_peak_2_1	1.180	1.058	**1.049**	1.139	1.039	**1.034**	1.14E+06	9.54E+05	**9.46E+05**
c_27_distorted_saddle_2_1	1.159	**1.041**	1.062	1.109	**1.030**	1.058	6.63E+08	**5.77E+08**	5.91E+08
c_29_stripes_2_1	1.142	**1.025**	1.036	1.112	**1.020**	1.029	4.02E+05	**3.60E+05**	3.65E+05
c_2_wave_2_1	1.129	1.046	**1.038**	1.117	1.022	**1.020**	4.05E+08	3.68E+08	**3.63E+08**
c_3_fold_2_1	1.097	1.081	**1.076**	1.070	**1.041**	1.042	5.12E+05	4.95E+05	**4.93E+05**
c_4_skew_raise_2_1	1.128	1.031	**1.024**	1.099	1.013	**1.012**	3.11E+08	2.67E+08	**2.65E+08**
c_6_polynomial_4_2	1.168	**1.022**	1.032	1.162	**1.012**	1.023	2.38E+09	**2.05E+09**	2.08E+09
c_9_broken_helix_2_1	1.220	1.046	**1.024**	1.164	1.037	**1.016**	7.02E+05	6.05E+05	**6.02E+05**
c_9_multi_dim_hook_4_2	1.153	**1.036**	1.043	1.132	**1.024**	1.031	2.98E+06	**2.83E+06**	2.85E+06
Avg. all datasets	1.176	1.049	1.047	1.134	1.028	1.029			

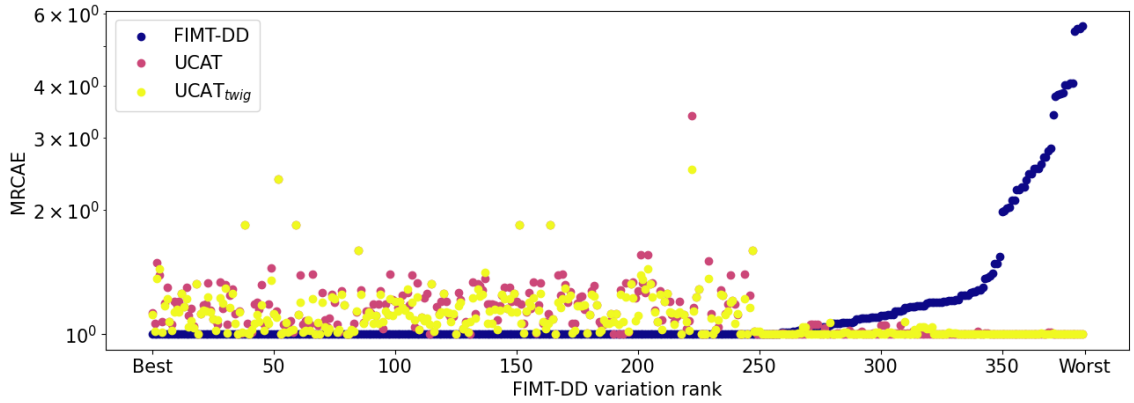

Figure 3: MRCAE results for the dataset c_13_multi_dim_splitter_tree_4_2. Each data point represents one variation of c_13_multi_dim_splitter_tree_4_2 for which the methods are evaluated. Experiments are sorted by the results of FIMT-DD for visual clarity.

We also reformulated other adaptive classification trees, like VFDTc [13], CVFDT [15], aCVFDT [22], HAT-ADWIN [3] and AHOT [4], for use in the regression context but found them not competitive with UCAT. One exception worth noting is AHOT, which exhibits superior performance over UCAT in specific scenarios where concept drift is considered independently, and data drift is disregarded as a potential disturbance. In such cases, where concept drift evolves gradually, it evades immediate detection by the active adaptive UCAT. AHOT excels particularly in scenarios featuring repeated drifts due to its rapid passive adaptation. However, in more complex environments characterized by infrequent drift occurrences, UCAT proves advantageous. Its selective approach to drift detection facilitates the acquisition of more sophisticated data structures. The structural differences between AHOT for regression and UCAT render further results of this comparison beyond the scope of this work. The variety of NSE provided by SiD^2Re enables a direct comparison of the methods for specific non-stationary properties like no drift, one/multiple concept drift and different noise levels. The averaged results over all datasets can be seen in Figure 4. These results are filtered based on some NSE properties found within the experiments. As we view the MRCAE the pairwise comparison between the selected properties have no meaning. The figure does not allow the conclusion that UCAT performs better in environments with drift than in environments without drift.

For stationary environments, i.e. no drift, FIMT-DD outperforms UCAT. UCAT's and $UCAT_{twig}$'s relative performance is the worst for stationary environments while FIMT-DD is at it's best. In contrast, FIMT-DD gets outperformed by UCAT and $UCAT_{twig}$ in NSE. For experiments with no or low levels of noise the relative performance remains the same as for NSE, but for experiments with high levels of noise the methods are less spread. As they all perform worse, the discrepancy between FIMT-DD and UCAT is reduced.

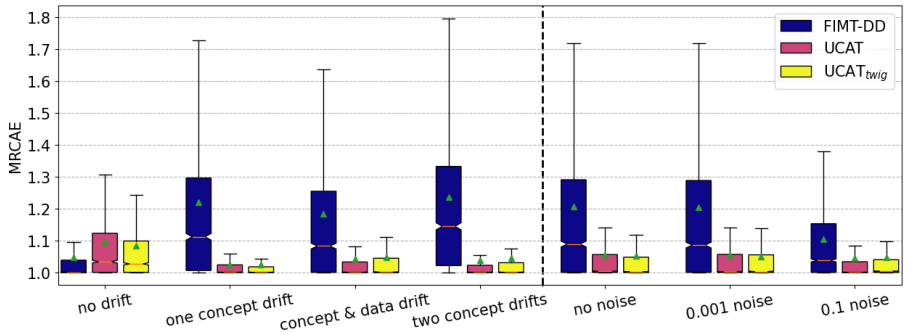

Figure 4: Summary of the average MRCAE performance of each method for NSE with different properties. The boxplot boxes show the interval between the inner two quartiles, while the whiskers reach ob to the last data point that falls into the 1.5 IQR. Outliers are excluded, the median is shown in orange and the mean is denoted by a green triangle.

Both trends can be traced back to the split deciders: Hoeffding's bound and U-Test. The U-Test requires more samples, leading to a slower growing tree. Slower growth limits performance on stationary data streams as shallow trees are very limited in their expressiveness. Conservative split decisions also slow down adaption in data drift scenarios, as the main priority in these cases is to restructure the data and not discard and reevaluate it.

In experiments with no or low levels of noise, as well as those involving data and/or concept drift, both variants of UCAT demonstrate significant improvements in CAE scores compared to FIMT-DD. This significance is determined using the U-Test to compare CAE performances, with a p-value of 0.05. However, in stationary or high noise environments, and when comparing UCAT to UCAT$_{\text{twig}}$, no significant differences are observed.

7 Conclusion

UCAT represents a novel incremental and adaptive tree learner for regression that uses the U-Test as the mathematical backbone for incremental growth and introduces new concepts for the adaption and pruning of trees: twig adaption and pruning by exposure. These additions ensure that the tree model is always supported by the most recent data. Our experiments show that the twig adaption can not provide significant performance gains. As of now, the twig adaption should not be used in applications without further tuning and runtime improvements. Beside the novelties introduced in UCAT the widely cited VFDT was the main point of inspiration together with the adaption methodology of FIMT-DD.

UCAT demonstrates competitive performance w.r.t. the most comparable and therefore most competing method FIMT-DD, proposed by Ikonomovska et al. [18]. Using UCAT instead of FIMT-DD on a variation in the benchmark results in an average reduction of the CAE by 13.33%. In drifting environments with no or low levels of noise, UCAT provides significant performance improvements over FIMT-DD.

The runtime of our method compared to that of FIMT-DD is about 43% to 193% slower, depending on whether twig adaption is used or not. While the method displays capabilities in terms of prediction performance and adaptability, the aspect of runtime efficiency remains an area of potential improvement and consideration for future work. Promising avenues for optimization lie in the two aspects that currently influence the runtime behavior of UCAT the most: The adaptation of twig thresholds and computation of the U-Test test statistic.

Looking forward, several opportunities arise for further investigation. The evaluation of different change detectors remains unexplored. PHT is predominantly used in the methods explored in this work but there is a plethora of different change detectors, many of them proposed in more recent years [23]. Additionally, the localization of drifts is a valuable property inherent to tree models and has not been systematically evaluated in this work. Drift localization describes the capabilities of an ML agent to identify drift and the source it originates from. This includes the possibility to extract information on what part of the data space might be adversely affected by drift and to focus adaption attempts on this locality.

References

[1] M. Baena-Garcıa, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, volume 6, pages 77–86, 2006.

[2] F. Bayram, B. S. Ahmed, and A. Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*, 245:108632, 2022.

[3] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *Advances in intelligent data analysis VIII*, volume 5772 of *Lecture Notes in Computer Science*, pages 249–260, Berlin, 2009. Springer.

[4] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 139–148, New York, NY, USA, 2009. ACM.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. Routledge, London, first edition edition, 1984.

[6] M. Ceci, A. Appice, and D. Malerba. Comparing simplification methods for model trees with regression and splitting nodes. In N. Zhong, Z. W. Raś, S. Tsumoto, and E. Suzuki, editors, *Foundations of Intelligent Systems*, pages 49–56, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[7] P. Chaudhuri, M.-C. Huang, W.-Y. Loh, and R. Yao. Piecewise-polynomial regression trees. *Statistica Sinica*, 4(1):143–167, 1994.

[8] D. M. Chickering, C. Meek, and R. Rounthwaite. Efficient determination of dynamic split points in a decision tree. In *Proceedings 2001 IEEE International Conference on Data Mining*. IEEE Comput. Soc, 2001.

[9] L. Da Xu and L. Duan. Big data for cyber physical systems in industry 4.0: a survey. *Enterprise Information Systems*, 13(2):148–169, 2019.

[10] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in Nonstationary Environments: A Survey. 10(4):12–25.

[11] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, New York, NY, USA, 2000. ACM.

[12] F. Enembreck, B. C. Ávila, E. E. Scalabrin, and J.-P. Barthès. Learning drifting negotiations. *Applied Artificial Intelligence*, 21(9):861–881, 2007.

[13] J. Gama, R. Fernandes, and R. Rocha. Decision trees for mining data streams. *Intelligent Data Analysis*, 10(1):23–45, 2006.

[14] C. Hasterok, J. Hermes, and B. Stratmann. Sid2re - a novel simulation framework for drifting regression data. In *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*, pages 1–8. IEEE, 2023.

[15] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2001. ACM.

[16] E. Ikonomovska, J. Gama, and S. Džeroski. Incremental multi-target model trees for data streams. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, New York, NY, USA, 2011. ACM.

[17] E. Ikonomovska, J. Gama, and S. Džeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, 2011.

[18] E. Ikonomovska, J. Gama, R. Sebastião, and D. Gjorgjevik. Regression trees from data streams with drift detection. In J. Gama, V. S. Costa, A. M. Jorge, and P. B. Brazdil, editors, *Discovery Science*, volume 5808 of *Lecture Notes in Computer Science*, pages 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[19] E. Ikonomovska, J. Gama, B. Zenko, and S. Dzeroski. Speeding-up hoeffding-based regression trees with options. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 537–544, New York, NY, USA, June 2011. ACM.

[20] D. Jayaratne, D. de Silva, D. Alahakoon, and X. Yu. Continuous detection of concept drift in industrial cyber-physical systems using closed loop incremental machine learning. *Discover Artificial Intelligence*, 1(1):1–13, 2021.

[21] Z. Kang, C. Catal, and B. Tekinerdogan. Machine learning applications in production lines: A systematic literature review. *Computers & Industrial Engineering*, 149:106773, 2020.

[22] J. Liu, X. Li, and W. Zhong. Ambiguous decision trees for mining concept-drifting data streams. *Pattern Recognition Letters*, 30(15):1347–1355, 2009.

[23] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, page 1, 2018.

[24] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[25] E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100, 1954.

[26] B. Pfahringer, G. Holmes, and R. Kirkby. New options for hoeffding trees. In M. A. Orgun, editor, *AI 2007*, Lecture notes in computer science Lecture notes in artificial intelligence, pages 90–99. Springer, Berlin, 2007.

[27] D. Potts. Incremental learning of linear model trees. In *Twenty-first international conference on Machine learning - ICML '04*, New York, New York, USA, 2004. ACM Press.

[28] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[29] L. Rutkowski, L. Pietruczuk, P. Duda, and M. Jaworski. Decision trees for mining data streams based on the mcdiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1272–1279, 2013.

[30] A. L. Suárez-Cetrulo, D. Quintana, and A. Cervantes. A survey on machine learning for recurring concept drifting data streams. *Expert Systems with Applications*, 213:118934, 2023.

[31] P. E. Utgoff. *Incremental induction of decision trees*. 1989.

[32] G. I. Webb, L. K. Lee, F. Petitjean, and B. Goethals. Understanding Concept Drift.

[33] F. Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics*, pages 196–202. Springer, New York, NY, 1992.

[34] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45, 2016.

A U-Test

The U-Test checks the equivalence of distribution functions underlying two populations of samples. This test does not take any other assumptions besides the ordinal properties of the samples in the populations X and Y. Regarding X and Y and their representing samples $x_i \in X$ and $y_j \in Y$ with $i \in \{0, \dots, n\} \subset \mathbb{N}$ and $j \in \{0, \dots, m\} \subset \mathbb{N}$ the test statistic is defined as follows:

$$U(X,Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} S\left(x_i, y_j\right) \quad S(x,y) = \begin{cases} 1, & \text{if } x > y \\ \frac{1}{2}, & \text{if } x = y \\ 0, & \text{if } x < y \end{cases} \tag{3}$$

For $m > 3, n > 3$ and $m + n > 19$ we can assume that U is distributed as follows:

$$U \sim N\left(\frac{mn}{2}; \frac{nm(n+m+1)}{12}\right) \tag{4}$$

Based on the value of the U test statistic (Equation 3) and the underlying distribution (Equation 4), we can determine the probability of a particular value to realize under the null hypothesis H_0 that both populations are drawn from the same underlying distribution. If this probability is smaller than a user-defined probability threshold $\delta \in [0, 1] \subset \mathbb{R}$, we reject the null hypothesis H_0 that both populations come from the same underlying distribution. For a given δ, the two-sided test checks if the test statistic falls within the tails of the normal distribution in Equation 4. The two-sided test fails if U (Equation 3) falls into $[-\infty, a]$ or $[b, \infty]$ where $\int_{-\infty}^{a} N\left(\frac{mn}{2}; \frac{nm(n+m+1)}{12}\right) = \delta/2$ and $\int_{b}^{\infty} N\left(\frac{mn}{2}; \frac{nm(n+m+1)}{12}\right) = \delta/2$. The one-sided test limits the alternate hypothesis to express that the first population is drawn to be ordinal larger (or smaller) than the second. This one-sided test fails if U (Equation3) falls into $[c, \infty]$ for $\int_{c}^{\infty} N\left(\frac{mn}{2}; \frac{nm(n+m+1)}{12}\right) = \delta$.

If we have to reject the null hypothesis of the one-sided test, we can conclude $P(\dot{x} < \dot{y}) > P(\dot{x} > \dot{y})$ or $P(\dot{x} < \dot{y}) < P(\dot{x} > \dot{y})$, depending on what side was tested for. $P(\dot{x} < \dot{y})$ is the probability that a random sample $\dot{x}$ drawn from the underlying distributions of population X is larger than a random sample $\dot{y}$ that is drawn from the underlying distributions of population Y.

B Page-Hinkley Test

The test's goal is to detect abrupt changes in the distribution parameters of an observed Gaussian signal. To achieve this, two variables are maintained and updated after each new measurement of the signal A that is monitored through measurements a_t.

$$M_T = \sum_{t=1}^{T} (a_t - \bar{a}_t - \delta) \qquad \delta \in \mathbb{R}_{>0} \tag{5}$$

$$\bar{a}_t = \frac{1}{t} \sum_{k=1}^{t} a_k$$

$$m_{|\mathcal{W}|} = \min_{T} \{M_T | T < |\mathcal{W}|\} \tag{6}$$

The value M_T (Equation 5) stores the historical deviation of the measured value a_t at time t from the accumulated mean over the last T samples. T can be equal to the number of samples seen or limit the analyzed number of samples to the $|\mathcal{W}|$ most recent ones. $m_{|\mathcal{W}|}$ (Equation 6) measures the minimal value of M_T seen for the last $|\mathcal{W}|$ samples, serving as a baseline to compare to.

PHT measures the difference between M_T and $m_{|\mathcal{W}|}$: $\mathrm{PHT}_{|\mathcal{W}|} = M_{T=|\mathcal{W}|} - m_{|\mathcal{W}|}$. If $PHT_{|\mathcal{W}|}$ exceeds a user-defined threshold $\lambda \in \mathbb{R}_{>0}$, a change is detected in the behavior of the monitored signal. λ as well as δ (Equation 5) are user-defined parameters of the method. Both parameters adjust the sensitivity of the test.

C Scaling

This section explores the theoretical and practical scaling of the algorithm in relation to the dataset size and the model parameters.

Notation

n total number of processed training samples

$|W|$ size of the active sliding window

D dimensionality of the input (number of features)

C.1 Runtime

The per-sample training cost can be decomposed into four steps:

1. **Tree traversal** – $O(\log |W|)$ (depth is logarithmic in $|W|$).
2. **Split–heuristic evaluation** – $O(|W|)$ (linear scan over the current window).
3. **U-test for split significance** – $O(|W| \log |W|)$ (sorting dominates).
4. **Least-squares refit after split/update** – $O(|W|D^2)$ (direct solver on $|W|$ samples with D features).

For all n samples the overall complexity is

$$O\big(n\,(|W| \log |W| + |W|D^2)\big).$$

C.2 Memory

Baseline implementation All past samples are retained, leading to a footprint of $O(n)$.

Projected optimisation Residual propagation stores the influence of past data in the internal-node parameters; therefore only the active window must be kept, reducing memory to $O(|W|)$.

Trade-offs

- Choosing $|W|$ too small speeds up training but may destabilise splits; an adaptive window is under investigation.

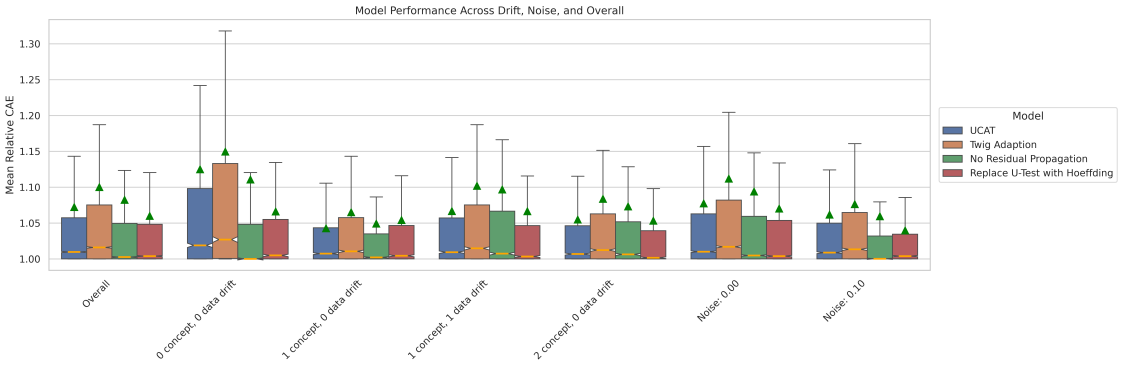

Figure 5: Summary of the average MRCAE performance for different UCAT variations for NSE with different properties. The boxplot boxes show the interval between the inner two quartiles, while the whiskers reach ob to the last data point that falls into the 1.5 IQR. Outliers are excluded, the median is shown in orange and the mean is denoted by a green triangle.

D Ablation Study on UCAT Components

To isolate the influence of the individual design choices we compare four variants of UCAT:

- the full version (**UCAT**),
- **UCAT$_{\text{twig}}$** – identical to UCAT but with twig–threshold adaptation enabled,
- **UCAT$_{\text{–res}}$** – residual propagation disabled,
- **UCAT$_{\text{–UTest}}$** – Hoeffding bound reintroduced instead of the proposed U-Test.

Table 2 reports the average relative errors (best method per stream = 1.0). Visualization of these results filtered based on stream properties can be seen in Figure 5.

Table 2: Relative CAE (best method per stream = 1.0).

Variant	mean	std	median	min	max
UCAT	1.072	0.232	1.010	1.000	4.904
UCAT$_{\text{twig}}$	1.100	0.267	1.016	1.000	4.904
UCAT$_{\text{–res}}$	1.082	0.255	1.003	1.000	4.903
UCAT$_{\text{–UTest}}$	**1.060**	**0.155**	**1.004**	1.000	3.186

Key findings

- **Split-decision rule dominates:** replacing the U-Test by the Hoeffding bound (UCAT$_{\text{–UTest}}$) lowers the *mean* relative CAE by $\approx 1\,\%$ and reduces the standard deviation from 0.232 to 0.155.
- **Twig adaptation is costly but rarely helps:** UCAT$_{\text{twig}}$ is $\approx 3\,\%$ worse than the base model and increases runtime by a factor of ~ 3.
- **Residual propagation has marginal effect:** turning it off (UCAT$_{\text{–res}}$) changes the mean error by $< 1\,\%$ while slightly widening the spread.

Recommendation For applications prioritising accuracy over theory, the simplest and most robust choice is **UCAT$_{\text{–UTest}}$** (i.e. UCAT with Hoeffding bound, no twig adaptation). When strict statistical guarantees are required, the full UCAT should be preferred, accepting a slight loss in average accuracy.

E Experiment Details

Table 3: Listing of what number of variations exhibit which property.

Characteristic	Values	Number of Variations
Drifts	no drift	1920
	concept drift	1920
	concept & data drift	1920
	two concept drifts	1920
Number of Datapoints	500	3840
	1000	3840
Noise Level	0	2560
	0.001	2560
	0.1	2560
Data Drift Type	faulty sensor	3840
	incremental	3840
Concept Drift Type	gradual	3840
	sudden	3840
Total Number of Variations		7680

Table 4: Parameter configurations for the bootstrap grid search used for parameter tuning.

	description	values	notation
variable parameters	samples between adaptions	50, 100	
	samples between leave model updates	10, 20	
	delta	0.01, 0.05, 0.1	δ
	window size	100, 200, 300	$\lvert\mathcal{W}\rvert$
general parameters	minimal number of samples for splits	20	
	minimal number of samples for leaves produced by a split	10	
	change test	PHT	
UCAT parameters	number of samples till twig adaption stops	50	a
	maximal twig adaption stepsizeas propotion of value range	0.2	$\hat{r}$

E.1 Hyperparameter Influences

- **min_split_samples**
  Minimum *total* number of observations that must be stored in a leaf before it is considered for a split test.
- **min_samples_produced_leaf**
  After a split each new child leaf must contain at least this many samples; otherwise the split is rejected.
- **n_samples_between_updates**
  Interval (in number of processed instances) at which a leaf re-evaluates its split candidates and refits its local model.

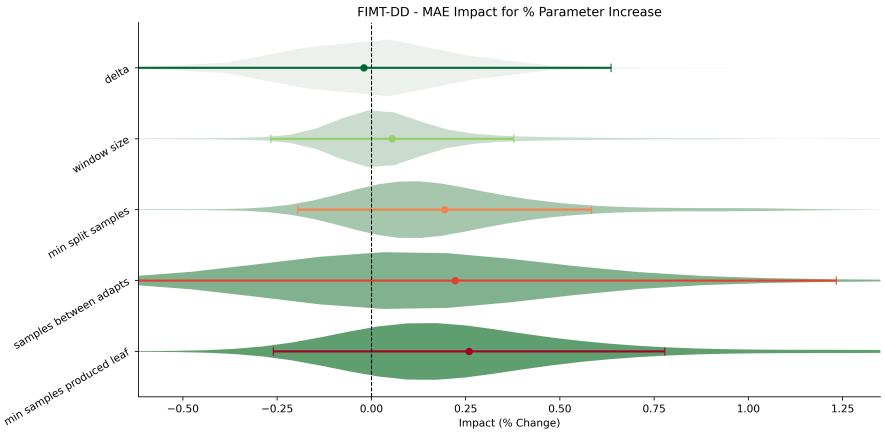

Figure 6: Parameter Dependencies of FIMT-DD: One unit of increase in a parameter mapped to the effects on prediction performance measure by MAE. The violin plots are capped to show the 1% to 99% range of values. The points show the average value and the wishker intervals show the 1 σ deviation.

- **samples_between_adapts**
  Interval that triggers the *adaptation routine* (drift check, rival branch growth, twig-threshold adjustment).
- **window_size** ($|W|$)
  Length of the sliding window that holds the most recent samples; all statistics, drift tests and pruning decisions are computed on this window.
- **delta**
  Confidence parameter of the split-significance test (U-Test in UCAT, Hoeffding bound in FIMT-DD); smaller values make the test more conservative.

Figures 6 and 7 show how an increase affects prediction performance measured on average measured by MAE and the Figures 8 and 9 show the effects on average runtime behavior. Green graphs indicate, that an increase in that variable is associated with improvements of the metric, while red values indicate, that an increase in that value in generally leads to an decrease. This only shows direct influences, but not more complex relations. For example: reducing the minimum number of samples required to establish a split usually leads to a faster runtime, as splits are done more rapidly. However, if we reduce this value below double the minimum number of samples that need to reside in a new leaf this parameter becomes completely irrelevant.

E.2 Computation Resources

The experiments are executed across a myriad of devices. On a device with a 13th Gen Intel(R) Core(TM) i7-1370P, 1900 MHz, 14 core processor and 64 GB of RAM clocked at 4800 MHz, an average experiment (evaluation of all methods on one variation) takes about 80 seconds and 250 MB of RAM capacity. The full set of experiments requires approximately 7.1 days if executed sequentially.

E.3 Asset Licenses

The primary external asset used is the SiD^{2}Re benchmark benchmark_datasets_v1.0_230623 which is licensed under the BSD-3-Clause and provided under `https://github.com/FraunhoferIOSB/SiD2Re`. Additional assets that where used for the implementation of UCAT and the evaluation are:

The provided implementation of UCAT is licensed under the BSD 3-Clause. The implementation will be further developed. If you are interested in an updated implementation or want to contribute please reach out to us. As of now the updated implementation will not be published in a public repository.

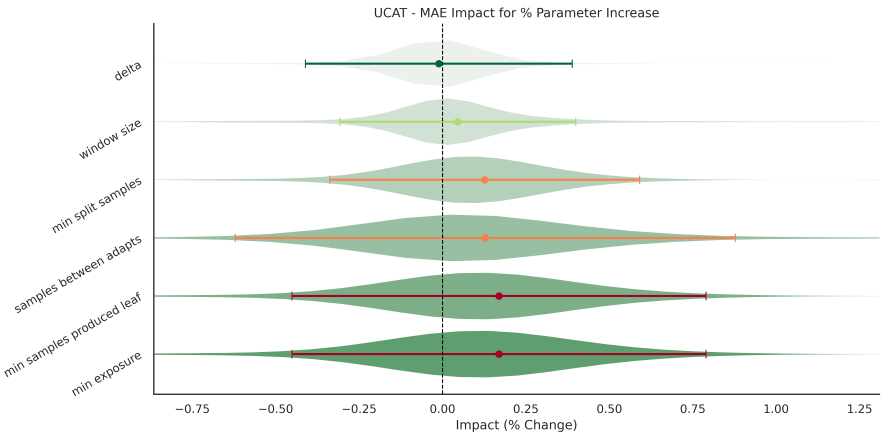

Figure 7: Parameter Dependencies of UCAT: One unit of increase in a parameter mapped to the effects on prediction performance measure by MAE. The violin plots are capped to show the 1% to 99% range of values. The points show the average value and the wishker intervals show the 1 σ deviation.

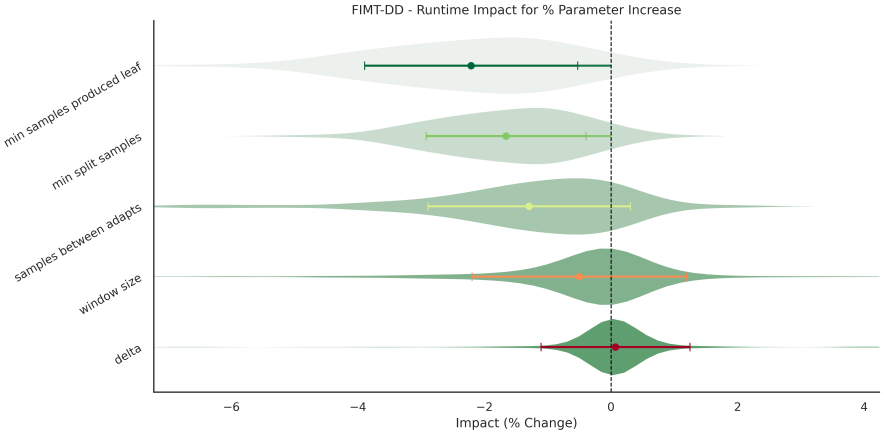

Figure 8: Parameter Dependencies of FIMT-DD: One unit of increase in a parameter mapped to the effects on runtime. The violin plots are capped to show the 1% to 99% range of values. The points show the average value and the wishker intervals show the 1 σ deviation.

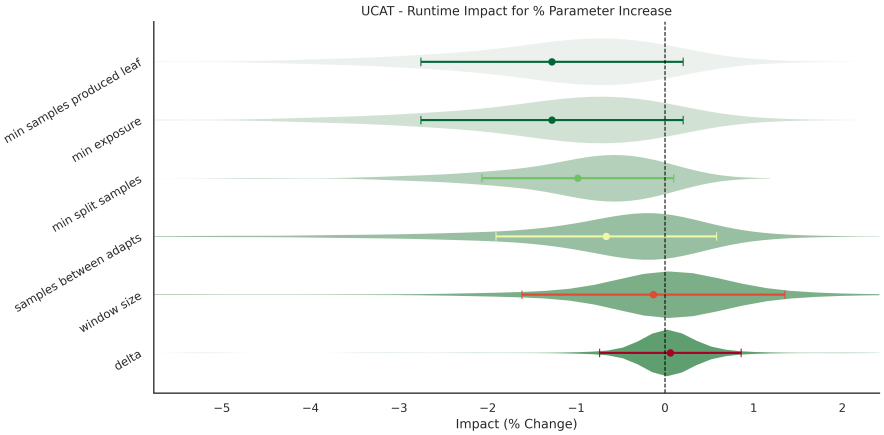

Figure 9: Parameter Dependencies of UCAT: One unit of increase in a parameter mapped to the effects on runtime. The violin plots are capped to show the 1% to 99% range of values. The points show the average value and the wishker intervals show the 1 σ deviation.

Table 5: Licenses and websites for used software

Software	**License**	**Website**
Python	Python Software Foundation License	`https://www.python.org`
Pandas	BSD 3-Clause License	`https://pandas.pydata.org`
NumPy	BSD 3-Clause License	`https://numpy.org`
PyArrow	Apache License 2.0	`https://arrow.apache.org`
tqdm	MIT License	`https://github.com/tqdm/tqdm`
scikit-learn	BSD 3-Clause License	`https://scikit-learn.org`
fire	Apache License 2.0	`https://github.com/google/python-fire`
grandalf	LGPL-3.0 License	`https://github.com/bdcht/grandalf`
numba	BSD 2-Clause License	`https://numba.pydata.org`
Poetry	MIT License	`https://python-poetry.org`