Appendix

A. Pseudocode

Algorithm 1 corresponds to Path A and B and Algorithm 2 corresponds to Path C in Fig. 5.

Algorithm 1: Proactive Drift Adaptation

```
Data: An instance from data stream
Input : ForegroundTrees, windows of each foreground tree's
        classification results W, DriftIntervals, AdaptationIntervals,
        w_adapt, ϵ_hybrid
Output: PotentialDriftedTrees, ActualDriftedTrees
PotentialDriftedTrees ← {};
ActualDriftedTrees ← {};
for i ∈ [0, NumOfForegroundTrees) do
    if DriftIntervals[i] > 0 then
        DriftIntervals[i] ← DriftIntervals[i] − 1;
        if DriftIntervals[i] == 0 then
            DriftIntervals[i] ← −1;
            PotentialDriftedTrees.append(ForegroundTrees[i]);
            AdaptationIntervals[i] ← w_adapt;
        end
    end
    if AdaptationIntervals[i] > 0 then
        AdaptationIntervals[i] ← AdaptationIntervals[i] − 1;
        if AdaptationIntervals[i] == 0 then
            AdaptationIntervals[i] ← −1;
            // Check for false positive
            if mean(W_1[i]) − mean(W_0[i]) > ϵ_hybrid then
                ActualDriftedTrees.append(ForegroundTrees[i]);
            end
        end
    end
end
```

B. Parameter Sensitivity Evaluations

There are four parameters that may have an effect on the performance of the Nacre framework, specifically the $w_{retrace}$ threshold, w_{adapt} for evaluating the proactively anticipated concepts, β threshold that controls the strictness for assessing if the proactively predicted drifts are false positives, $\delta_{stability}$ for detecting stability of the trees. Table IV shows the impact of the parameters is relatively stable, and that Nacre is robust under a controlled environment. These parameters do not have statistically significant impact on performance, and can be fine tuned for risk-averse environments.

The higher cumulative gains come from the proactively correctly selected candidate trees. The anticipated drifted trees must be accurate for the base recurrent drift classifier, PEARL, to find the correct candidate trees that fit the incoming concepts. Otherwise, the candidate tree pool gets filled with wrong trees, since the size of the candidate trees is fixed and is evicted by First In, First Out. This will generate more noise and deteriorate performance. The stable results show that Nacre can accurately anticipate the correct drifted trees, which generates further accuracy gains.

Table V investigates the impact of the β parameter. When the β values are varied the cumulative accuracy gains may be accrued from both the True Positive (TP) and False Positive (FP) signals that a foreground tree needs to be replaced with

Algorithm 2: Proactive Drift Prediction

```
Data: Instance x from data stream
Input : ForegroundTrees, windows of each foreground tree's
        classification results W, LastDriftedTrees,
        DriftIntervalSequences, DriftSequencePredictors, Clusterers,
        Data Buffer maintaining a window of the data, current
        timestamp t_current, w_retrace, ϵ_stable
Output: DriftIntervals
for i ∈ [0, NumOfForegroundTrees) do
    if not IsStable(i) then
        continue;
    end
    t ← FindLastActualDriftPoint(i);
    DriftIntervals[i] ← PredictNextDriftInterval(i, t);
end
function IsStable(i):
    if ForegroundTrees[i].predict(x) == x.label then
        W[i].push(0) ;
    else
        W[i].push(1) ;
    end
    if mean(W_1[i]) − mean(W_0[i]) > ϵ_stable then
        while mean(W_1[i]) − mean(W_0[i]) > ϵ_stable do
            W[i].pop();
        end
        return True;
    end
    return False;
end
function FindLastActualDriftPoint(i):
    CurrentTreeWindow ← {};
    DriftedTreeWindow ← {};
    t ← t_current;
    while x_t in Data Buffer do
        if ForegroundTrees[i].predict(x) == x_t.label then
            CurrentTreeWindow.push(1);
        else
            CurrentTreeWindow.push(0);
        end
        if LastDriftedTrees[i].predict(x) == x_t.label then
            DriftedTreeWindow.push(1);
        else
            DriftedTreeWindow.push(0);
        end
        if Size(CurrentTreeWindow) > w_retrace then
            CurrentTreeWindow.pop();
            DriftedTreeWindow.pop();
            if Sum(DriftedTreeWindow) >
              Sum(CurrentTreeWindow) then
                break;
            end
        end
        t ← t − 1;
    end
    return t;
end
function PredictNextDriftInterval(i, t):
    LastDriftInterval ← t_current − t ;
    LastDriftInterval ←
      Clusterers[i].TrainAndPredict(LastDriftInterval);
    DriftIntervalSequences[i].push(LastDriftInterval);
    DriftSequencePredictors[i].train();
    DriftIntervalSequences[i].pop();
    return
      DriftSequencePredictors[i].predict(DriftIntervalSequences[i]);
end
```

TABLE IV: Parameter Sensitivity Evaluation

Agrawal Poisson 10 with Abrupt Drifts				
$w_{retrace}$	Acc. Gain per Drift (%)	Cum. Acc. Gain (%)	Runtime (min)	#Trees
25	8.67 ± 0.71	5253.54 ± 476.62	10.93 ± 0.28	4525 ± 75
50	8.57 ± 0.76	5265.40 ± 329.91	11.91 ± 0.38	4463 ± 86
75	8.56 ± 0.76	5169.56 ± 625.20	12.28 ± 0.26	4457 ± 95
100	8.82 ± 0.68	5421.69 ± 359.90	12.37 ± 0.27	4436 ± 85
500	8.63 ± 0.71	5220.61 ± 508.49	12.83 ± 0.28	4256 ± 194
1000	8.53 ± 0.94	5223.94 ± 611.42	13.21 ± 0.50	4243 ± 177
w_{adapt}	Acc. Gain per Drift (%)	Cum. Acc. Gain (%)	Runtime (min)	#Trees
200	8.67 ± 0.71	5253.54 ± 476.62	10.93 ± 0.28	4525 ± 75
400	8.82 ± 0.52	5407.84 ± 425.27	10.94 ± 0.46	4527 ± 78
600	8.79 ± 0.82	5298.67 ± 459.77	10.60 ± 0.19	4285 ± 144
800	8.62 ± 0.70	5183.53 ± 413.82	10.63 ± 0.28	4302 ± 165
1000	8.27 ± 0.78	4967.52 ± 370.72	10.69 ± 0.27	4273 ± 153
$\delta_{stability}$	Acc. Gain per Drift (%)	Cum. Acc. Gain (%)	Runtime (min)	#Trees
0.1	8.57 ± 0.56	5293.45 ± 406.84	11.26 ± 0.45	4484 ± 93
0.01	8.67 ± 0.71	5253.54 ± 476.62	10.93 ± 0.28	4525 ± 75
0.001	8.57 ± 0.42	5260.79 ± 480.39	10.61 ± 0.34	4461 ± 106
β	Acc. Gain per Drift (%)	Cum. Acc. Gain (%)	Runtime (min)	#Trees
0.1	8.72 ± 0.54	5313.09 ± 345.13	11.06 ± 0.30	4353 ± 91
0.3	8.63 ± 0.63	5269.30 ± 396.78	11.07 ± 0.27	4349 ± 104
0.5	8.68 ± 0.50	5223.91 ± 437.61	11.05 ± 0.32	4376 ± 76
0.7	8.45 ± 0.54	5085.69 ± 359.44	11.04 ± 0.22	4356 ± 85
0.9	8.67 ± 0.71	5253.54 ± 476.62	10.93 ± 0.28	4525 ± 75
Agrawal Poisson 10 with Gradual Drifts				
$w_{retrace}$	Acc. Gain per Drift (%)	Cum. Acc. Gain (%)	Runtime (min)	#Trees
25	5.10 ± 1.26	2580.51 ± 704.09	10.48 ± 0.35	4046 ± 187
50	5.09 ± 0.90	2685.97 ± 479.50	11.61 ± 0.54	4042 ± 213
75	5.35 ± 0.77	2986.42 ± 460.61	11.83 ± 0.42	3998 ± 203
100	5.30 ± 0.94	2817.13 ± 465.44	11.95 ± 0.53	4012 ± 210
w_{adapt}	Acc. Gain per Drift (%)	Cum. Acc. Gain (%)	Runtime (min)	#Trees
200	5.31 ± 1.26	2820.86 ± 629.15	10.53 ± 0.35	4030 ± 224
400	5.10 ± 1.26	2580.51 ± 704.09	10.48 ± 0.35	4046 ± 187
600	5.18 ± 1.32	2697.27 ± 638.75	10.48 ± 0.31	4041 ± 190
800	5.46 ± 0.91	3004.78 ± 448.53	10.55 ± 0.38	4044 ± 198
1000	5.31 ± 0.79	2878.00 ± 355.03	10.65 ± 0.29	4012 ± 225
$\delta_{stability}$	Acc. Gain per Drift (%)	Cum. Acc. Gain (%)	Runtime (min)	#Trees
0.1	5.30 ± 0.97	2846.36 ± 456.47	10.73 ± 0.35	4044 ± 193
0.01	5.10 ± 1.26	2580.51 ± 704.09	10.40 ± 0.52	4046 ± 187
0.001	5.26 ± 0.86	2909.87 ± 505.45	10.42 ± 0.42	4038 ± 208
β	Acc. Gain per Drift (%)	Cum. Acc. Gain (%)	Runtime (min)	#Trees
0.1	4.75 ± 0.75	2522.29 ± 373.67	10.75 ± 0.44	3958 ± 197
0.3	4.88 ± 0.91	2614.09 ± 440.95	10.76 ± 0.39	3951 ± 187
0.5	4.55 ± 1.03	2408.31 ± 400.21	10.76 ± 0.46	3963 ± 196
0.7	4.81 ± 0.86	2594.65 ± 441.90	10.67 ± 0.36	3939 ± 204
0.9	5.10 ± 1.26	2580.51 ± 704.09	10.40 ± 0.52	4046 ± 187

a candidate tree. Even if a candidate tree is wrongly signalled as a false positive, it gets proactively selected from the online tree repository and stored in the candidate tree pool. When this occurs, the tree is evaluated for a longer period of time and is more likely to be swapped into the foreground tree group when a drift detector is later triggered.

TABLE V: TP and FP values based on varying β on Agrawal Poisson 10 with abrupt drifts

β	TP	FP
0.1	0 ± 0	838 ± 628
0.5	4 ± 4	831 ± 620
0.7	29 ± 25	806 ± 600
0.9	214 ± 163	650 ± 479

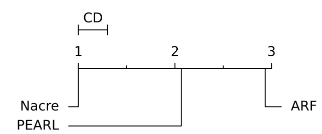
1) Real-world datasets: Table VI shows β value has a higher impact on the cumulative accuracy gain on a long running stream, thus this parameter needs to be fine tuned to obtain the optimal accuracy gains.

TABLE VI: Results for Sensor dataset based on varying β

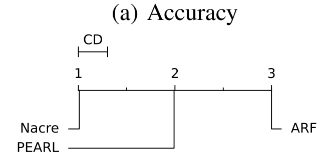
β	Acc (%)	Kappa (%)	Cum. Acc. (%)	Runtime (min)	#Trees
0.1	34.34 ± 15.61	32.19 ± 16.34	31455	32.96	485
0.3	34.69 ± 15.88	32.58 ± 16.57	32221	23.37	484
0.5	35.34 ± 16.97	33.22 ± 17.70	33656	33.15	499
0.7	33.70 ± 16.16	31.53 ± 16.93	30021	23.78	478
0.9	34.69 ± 17.00	32.54 ± 17.77	32234	32.63	503

C. Critical Difference Diagrams

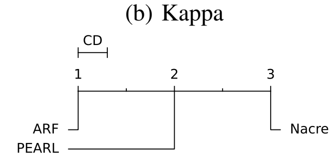
The critical difference diagrams for Table I are given in Fig. 9. The diagrams are ranked from highest to lowest for accuracy and kappa, and lowest to highest for runtime, so high rankings are preferable.



(a) Accuracy



(b) Kappa



(c) Runtime

Fig. 9: Critical difference diagrams from posthoc Nemenyi tests of rankings for synthetic datasets.