

Run-Time Adaptation of Complex Event Forecasting

Manolis Pitsikalis

NCSR Demokritos

Athens, Greece

manospits@iit.demokritos.gr

Elias Alevizos

NCSR Demokritos

Athens, Greece

The American College of Greece

Athens, Greece

alevizos.elias@iit.demokritos.gr

Nikos Giatrakos

Technical University of Crete

Chania, Greece

ngiatrakos@tuc.gr

Alexander Artikis

NCSR Demokritos

Athens, Greece

University of Piraeus

Piraeus, Greece

a.artikis@iit.demokritos.gr

Abstract

Complex Event Forecasting (CEF) is a process whereby complex events of interest are forecast over a stream of simple events. CEF facilitates proactive measures by anticipating the occurrence of complex events. This proactive property, makes CEF a crucial task in many domains; for instance, in maritime situational awareness, forecasting the arrival of vessels at ports allows for better resource management, and higher operational efficiency. However, our world's dynamic and evolving conditions necessitate the use of adaptive methods. For example, for safety reasons, maritime vessels may adapt their routes to avoid powerful swell waves; in fraud analytics, fraudsters evolve their tactics to avoid detection etc. CEF systems typically rely on probabilistic models, trained on historical data. This renders such CEF systems inherently susceptible to data evolutions that can invalidate their underlying models. To address this problem, we propose RTCEF, a novel framework for Run-Time Adaptation of CEF, based on a distributed, service-oriented architecture. We evaluate RTCEF on two use-cases and our reproducible results show that our proposed approach has significant benefits in terms of forecasting performance without sacrificing efficiency.

CCS Concepts

- **Computer systems organization** → **Real-time systems**;
- **Theory of computation** → *Formal languages and automata theory*;
- **Mathematics of computing** → *Bayesian computation*.



This work is licensed under a Creative Commons Attribution 4.0 International License.

DEBS '25, Gothenburg, Sweden

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1332-3/25/06

<https://doi.org/10.1145/3701717.3730539>

Keywords

complex event forecasting, run-time adaptation, optimisation

ACM Reference Format:

Manolis Pitsikalis, Elias Alevizos, Nikos Giatrakos, and Alexander Artikis. 2025. Run-Time Adaptation of Complex Event Forecasting. In *The 19th ACM International Conference on Distributed and Event-based Systems (DEBS '25)*, June 10–13, 2025, Gothenburg, Sweden. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3701717.3730539>

1 Introduction

Complex Event Forecasting (CEF) is akin to Complex Event Recognition (CER) [17, 22], but with a forward-looking perspective. Both tasks operate on a stream of simple events, while their output consists of Complex Events (CEs). For example, in a maritime situational awareness [4], the stream of simple events would contain positional messages of vessels, while the output stream would contain maritime CEs such as (illegal) fishing activities. The difference between CER and CEF is that, in the former, elements of the output stream refer to CE detections, while in CEF, elements of the output stream refer to the probability of a CE happening in the future. Consequently, CER enables reactive responses upon CE detections, while CEF supports proactive measures by anticipating future CEs. This proactive property renders CEF systems highly desirable. CER and CEF applications span diverse domains, such as maritime situational awareness [2, 28] whereby CEs such as fishing are detected or forecast over a stream of maritime data; credit card fraud management [2, 32] whereby frauds are detected or forecast over a stream of transaction data; and so on.

CEF operates over constantly evolving conditions. Take for example the problem of maritime route optimisation. Vessels may follow a different route depending on the swell wave conditions, i.e., waves that can significantly affect navigation and vessel stability [11]. Another example is financial fraud

detection—fraudsters constantly adapt their tactics to avoid getting caught [3]. Moreover, CEF systems rely on probabilistic models trained on historical data [1, 25, 26]. This renders CEF systems inherently susceptible to evolutions in the input that can invalidate their underlying models—recall the previous example relating maritime routes with weather. Additionally, as with the majority of trainable models, CEF models have hyperparameters that require fine tuning for optimal performance. Wayeb [2], a state-of-the-art CEF engine, is no exception to the above.

To address the above challenges we propose RTCEF, an open-source framework for Run-Time Adaptation of CEF over constantly evolving data streams. RTCEF adopts a distributed architecture comprising targeted services to effectively (a) enable run-time update of CEF models with little to no downtime, (b) ensure that transition between models does not cause loss of forecasts. In other words, RTCEF supports continuous adaptation to dynamic changes in the input stream with little to no effect on efficiency. Furthermore, RTCEF provides a trend-based policy which acts as a decision making mechanism to distinguish whether hyperparameter optimisation or CEF model retraining, without changing hyperparameters, is the best way to maintain accurate forecasts. In addition to RTCEF, we also present offCEF, a baseline framework for CEF hyperparameter optimisation, that in contrast to RTCEF, optimises CEF in an offline manner. Our contributions are:

- we introduce RTCEF, an open-source¹ framework addressing the challenges and requirements of run-time CEF over evolving data streams through a distributed architecture which enables CEF to seamlessly run on par with training or optimisation tasks, ensuring no disruptions;
- we formally prove that RTCEF achieves lossless runtime adaptation, i.e., no forecast is lost upon hyperparameter optimisation or retraining decisions;
- we extensively evaluate RTCEF and offCEF on two real-world critical use-cases from the maritime and financial domains and our *reproducible* results validate that RTCEF, compared to offCEF, can significantly improve forecasting performance with little to no lag upon run-time changes.

2 Background

CEF is a task that allows forecasting CEs of interest, such as fishing activities or vessel rendezvous, over an input stream of simple events; e.g., timestamped position messages of maritime vessels. Forecasts involve the occurrence of a CE in the future accompanied by a degree of certainty [2]. This behaviour is usually derived from stochastic models that project into the future evolutions of the input that can cause a detection of a CE. For the task of CEF, we utilise Wayeb,

Table 1: An example stream for a single vessel composed of five events. Each event has a vessel identifier, a value for that vessel’s speed and a timestamp.

vessel ID	78986	78986	78986	78986	78986	...
speed	5	3	9	14	11	...
timestamp	1	2	3	4	5	...

a CEF engine introduced in [2], which employs symbolic automata as its computational model. The user submits a query/pattern to Wayeb which is then compiled into a symbolic, streaming automaton. This automaton may be used to perform event recognition, i.e., to detect instances of pattern satisfaction upon a stream of input events. Whenever the automaton reaches a final state, a complex event is reported as having occurred. In order to perform forecasting, Wayeb constructs a probabilistic model of the compiled automaton, by using part(s) of a stream for training. The model allows us to infer, at any given moment, the possible paths that the automaton may follow in the future. By searching among the possible future paths, we can estimate when the automaton is expected to reach a final state and thus report a CE. The output of Wayeb thus consists of two streams: a) one reporting the detected events, and b) one reporting the forecasts of events expected to occur in the future.

Wayeb has clear, compositional semantics for the patterns expressed in its language and can support most of the common operators [17]. Wayeb’s patterns are expressed as Symbolic Regular Expressions (*SREs*), where terminal expressions are Boolean expressions, i.e., logical formulae that use the standard Boolean connectives of conjunction ‘ \wedge ’, disjunction ‘ \vee ’ and negation ‘ \neg ’ on predicates [2]. Wayeb *SREs* are defined using the grammar below:

$$\begin{aligned}
 R ::= & R_1 + R_2 \text{ (union)} \mid R_1 \cdot R_2 \text{ (concatenation)} \\
 & \mid R_1^* \text{ (Kleene-star)} \mid !R_1 \text{ (complement)} \\
 & \mid \psi \text{ (Boolean expression)}
 \end{aligned}$$

R_1, R_2 are regular expressions, and ψ is a Boolean expression. The semantics of the above operators are detailed in [2]. Evaluation of *SREs* on a stream of events requires first their compilation into symbolic automata. Transitions in symbolic automata are labeled with Boolean expressions. For a symbolic automaton to move to another state, it first applies the Boolean expressions of its current state’s outgoing transitions to the element last read from the stream. If an expression is satisfied, then the corresponding transition is triggered and the automaton moves to that transition’s target state. For example, in maritime situational awareness, a domain expert could use Wayeb’s language to specify a pattern $R := (speed > 10) \cdot (speed > 10)$ for identifying

¹<https://zenodo.org/records/15229227>

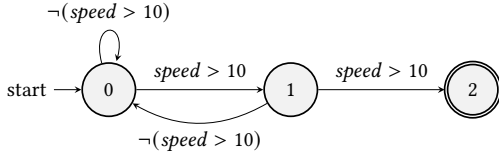


Figure 1: Streaming symbolic automaton created from the expression $R := (speed > 10) \cdot (speed > 10)$.

speed violations in specific areas where the maximum allowed speed is 10 *knots*. This pattern is satisfied when there are two consecutive events where a vessel’s speed exceeds the threshold. The compiled automaton corresponding to R is illustrated in Figure 1. For an input stream consisting of the events in Table 1, the automaton would run as follows. For the first three input events, the automaton remains in state 0. After the fourth event, it moves to state 1 and after the fifth event it reaches its final state, state 2, triggering also a CE detection for R at *timestamp* = 5.

To perform CEF, Wayeb needs a probabilistic description for a symbolic automaton derived from a *SRE*. For this purpose, Wayeb employs Prediction Suffix Trees (PSTs) [30, 31]—a form of Variable-order Markov Models. Variable-order Markov Models, compared to fixed-order Markov models, capture longer-term dependencies as in practice they allow for higher order (m) values than the latter. Each node in a PST contains a “context” and a distribution that indicates the probability of encountering a symbol, conditioned on the context. Figure 4 (top left) shows an example of a PST. Each “symbol” of a PST corresponds to a predicate of the automaton for which we want to build a probabilistic model. For example, the predicate $(speed > 10)$ may be such a “symbol” for the pattern R . The same predicate, but negated i.e., $\neg(speed > 10)$, may be another such “symbol”. Learning a PST from data is an incremental process that adds new nodes corresponding to symbols only when necessary [2, 30, 32]. The learning process involves two key hyperparameters. First, the $pMin \in [0, 1]$ hyper-parameter which corresponds to a threshold determining which symbols are deemed to be “too rare” to be taken under consideration by the learning algorithm (symbols with a probability of appearance less than $pMin$ are discarded). Second, the γ hyperparameter is a symbol distribution smoothing parameter.

With the resulting PST, for every state q of an automaton and the last m (order of the PST) symbols of the input stream, we can calculate the waiting-time distribution (W_q), that is, the probability of reaching a final state in n transitions from a state q . Recall that a CE is detected whenever an automaton reaches a final state. Figure 4 (middle and bottom left) shows an example of an automaton and the waiting-time distributions learnt from a training dataset. Wayeb then performs CEF as follows. Given the current state q of an automaton,

using W_q , we compute the probability of reaching a final state (p_{CE}) within the next n transitions (or, equivalently, input events). If p_{CE} exceeds a confidence threshold $\theta_{fc} \in [0, 1]$, Wayeb emits a “positive” forecast (denoting that the CE is expected to occur), otherwise a “negative forecast” (no CE is expected) is emitted.

A forecast for a CE is characterised as a True Positive (*TP*) if a positive forecast (i.e., the CE will occur in the future) was emitted and the CE indeed occurred or, respectively, as a False Positive (*FP*) if the CE did not occur. A forecast for a CE is characterised as a True Negative (*TN*) if a negative forecast is emitted (i.e., the CE will not occur in the future) and the CE does not occur or, respectively, as a False Negative (*FN*) if the CE does occur. Note that a forecast cannot be evaluated as *TP*, *FP*, *TN* or *FN* upon its emission. It can be evaluated as such after the next n input events have arrived, at which point we can know whether the forecast event did occur or not. Given that Wayeb performs both CEF and CER, forecasts are evaluated on-the-fly. Using these classifications of forecasts, the performance of CEF may be quantified through Matthew’s Correlation Coefficient (*MCC*), defined as follows:

$$MCC = \sqrt{Precision \times Recall \times Specificity \times NPV} - \sqrt{FDR \times FNR \times FPR \times FOMR} \quad (1)$$

where $NPV = \frac{TN}{TN+FN}$, $Specificity = \frac{TN}{TN+FP}$, $FDR = 1 - Precision$, $FNR = 1 - Recall$, $FPR = 1 - Specificity$ and $FOMR = 1 - NPV$. *Precision* and *Recall* are defined as usual. Therefore, $MCC \in [-1, 1]$ estimates the agreement, in which case $MCC = 1$, (or disagreement, resp. $MCC = -1$) between the emitted forecasts and observations. In contrast to F1-Score, which takes into account only positive instances, *MCC* takes into account both positive and negative instances. Since Wayeb produces both positive and negative forecasts, *MCC* is a fitting choice.

Given the above, the hyperparameters required for training Wayeb models, i.e., PSTs, are the following. The maximum order m of the PST, along with the symbol retaining probability threshold $pMin$, the symbol distribution smoothing parameter γ and the confidence threshold θ_{fc} . The naive way to train a Wayeb PST is to manually fix the values of these hyperparameters and then select a training dataset from which a PST may be extracted. This process can be performed offline and Wayeb may then employ the learnt PST for online event forecasting. As we explain below, this is not the proper way to go.

3 Challenges of CEF

Performing CEF over constantly evolving data streams exhibits several significant challenges.

Challenge 1. *CEF hyperparameter optimisation entails complicated trade-offs.*

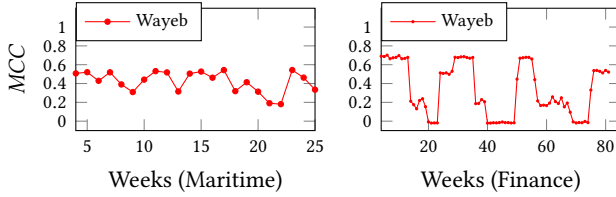


Figure 2: MCC scores of Wayeb for forecasting a CE related to the arrival of vessels at a port over maritime positional data (left), and forecasting the occurrence of financial frauds over transactional data (right).

In Wayeb, although setting the maximum order m generally improves accuracy, it leads to longer training times. Similarly, finding the optimal value for θ_{fc} , i.e., the probability threshold for emitting a forecast, is a crucial step as overly low or high θ_{fc} values can cause many false positives or false negatives, respectively. Furthermore, $pMin$, the threshold determining which symbols are “too rare” to be included during training, can also affect accuracy. High $pMin$ values can produce simpler models but may discard useful symbols. On the other hand, excessively low values of $pMin$ can degrade the accuracy of the forecasts due to overfitting of the PSTs to insignificant symbols. The symbol distribution smoothing parameter γ behaves in the same manner. Manually fixing the above combination of parameters would lead to sub-optimal results. Moreover, exhaustive hyperparameter space exploration is of high computational complexity making it prohibitive for run-time settings where Wayeb’s hyperparameters need to be tuned multiple times to adjust to data evolutions that invalidate the deployed PSTs. To address these issues, we propose RTCEF, a framework for the run-time adaptation of CEF. RTCEF, presented in the following section, employs Bayesian optimisation to efficiently explore only a small fraction of the parameter space and learn the optimal combination of hyperparameter values for the entire parameter space. Furthermore, in contrast to traditional Bayesian optimisation setups, we do not start each optimisation run from scratch, instead we leverage knowledge from previous runs by constantly refreshing a sample set with new samples.

Challenge 2. *Run-time CEF optimisation has inherently increased complexity, while CEF applications typically involve processing Big streaming Data with volatile statistical properties that can severely affect CEF performance.*

Run-time optimisation of Wayeb is a complicated and challenging task because it involves (a) training a Variable-order Markov Model, that is, a PST (b) using it for estimating waiting-time distributions and (c) subsequently performing probabilistic, automaton-based pattern matching (Figure 4). Consequently, employing an analytical formula to model the performance of Wayeb for a given hyperparameter set and input is impossible without training and testing Wayeb.

Although an initial optimal hyperparameter set can be found for some historical dataset, in the run-time settings environmental changes might happen that can invalidate the deployed Wayeb’s PST. See for example Figure 2, which shows the MCC score of Wayeb on forecasting two CE in a maritime situational awareness setting and a financial fraud detection setting. In both cases data evolutions in the input stream cause significant fluctuations and drops in CEF scores. Consequently, there is a need for continuous adaptation over evolving data streams. RTCEF addresses this challenge with run-time PST retraining or hyperparameter optimisation.

Challenge 3. *Time-critical applications employing CEF require undisrupted production of forecasts.*

In critical applications, such as maritime situational awareness or credit card fraud management, updating the currently deployed PST with a new version should not stall the production of CE forecasts, as such delays would halt the proactive decision-making mechanisms of stakeholders. Consequently, updating the deployed PST with newly revised versions should happen in negligible time ensuring no disruptions in CEF and no loss of forecasts. Finally, although hyperparameter optimisation can result in high performing models, it does not come without a cost. Hyperparameter optimisation is, resource-wise, an expensive procedure which should only happen when necessary. The RTCEF framework addresses the above challenges using a novel distributed, service-oriented architecture.

4 Run-Time CEF Adaptation

We start by presenting offCEF, a baseline framework for hyperparameter optimisation of CEF under the stationarity assumption, i.e., assuming that there are no evolutions in the input that might invalidate the CEF model. Subsequently, we present RTCEF, which addresses all challenges of run-time CEF mentioned in Section 3.

4.1 CEF Under the Stationarity Assumption

Under the stationarity assumption, a single PST, produced through training on some historical, static dataset, will suffice for future input. Consequently, in this setting, we may use a framework for offline hyperparameter optimisation, hereafter offCEF. The aim of offCEF is the identification of an optimal configuration c_{opt} that yields the best performance for Wayeb, quantified by the MCC score (see Equation (1)). A configuration c is defined as follows:

$$c = [m, \theta_{fc}, pMin, \gamma]$$

where m , θ_{fc} , $pMin$ and γ are Wayeb’s hyperparameters (see Section 2) with their domain empirically set as:

$$\begin{array}{l|l} m \in [1, 5] & \theta_{fc} \in [0.0, 1.0] \\ pMin \in [0.0001, 0.01] & \gamma \in [0.0001, 0.01] \end{array}$$

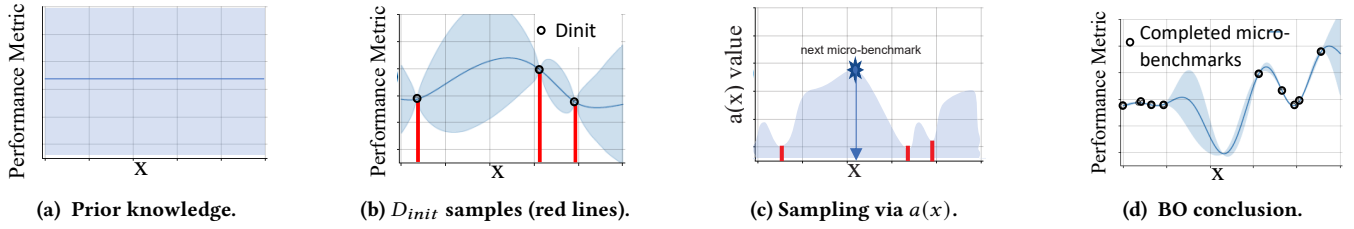


Figure 3: Bayesian Optimisation Operation.

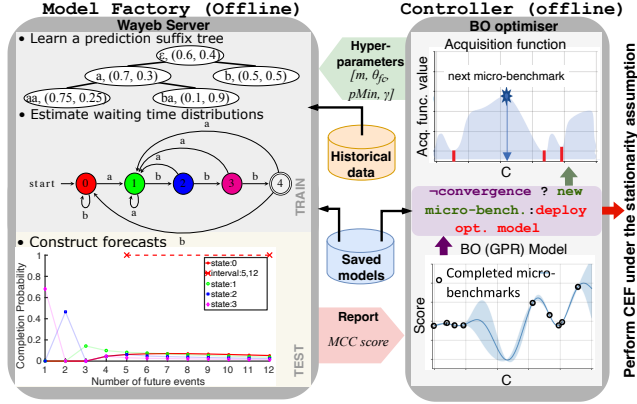


Figure 4: Architecture of offCEF.

Given the infinite parameter combinations, exhaustive search is computationally prohibitive. Furthermore, due to Wayeb’s complexity, performance for a given parameter set cannot be known beforehand. Consequently, to find the optimal configuration c we employ Bayesian optimisation (BO) [6, 14] i.e., a stochastic method for optimising expensive-to-evaluate objective functions that are complex or cannot be described by analytic formulae. In our work, the objective function is defined as $f(c) = MCC_c$, where MCC_c denotes the MCC score of Wayeb given configuration c .

The goal of BO is to find the vector of Wayeb’s hyperparameters that maximises CEF performance, using a minimal set of Wayeb training-test runs, termed ‘micro-benchmarks’, as training samples. Unlike other optimisation methods [34] BO does not require a high number of micro-benchmarks or an analytical formula [6, 14, 32]. BO employs a probabilistic model—called surrogate model—to approximate the unknown objective function, in our case CEF performance quantified by MCC, and iteratively refines this model. We employ a Gaussian Process Regressor (GPR) as the surrogate model. Initial beliefs about the objective function must be formulated before observing any data. In BO, priors are often specified for the mean and covariance functions of the Gaussian Process model. For example, a prior belief might suggest that the function is smooth and lies within a certain range of values. Priors are represented as:

$$f(c) \sim \text{GP}(\mu_0(c), k_0(c, c'))$$

where $\mu_0(c)$ and $k_0(c, c')$ are the prior mean and covariance (kernel) functions, respectively.

Every time we observe a new micro-benchmark and collect CEF performance metrics by training and testing Wayeb given a configuration c , we acquire a new training sample (c, MCC_c) , to fit on the GPR, thereby updating our posterior belief in light of new evidence. The posterior distribution represents our updated knowledge about Wayeb’s performance and after observing n new training samples, denoted by $Data$, the posterior is given by:

$$f(c) \mid Data \sim \text{GP}(\mu_n(c), k_n(c, c'))$$

$\mu_n(c)$ and $k_n(c, c')$ being the posterior mean and covariance functions updated through Bayesian inference [6, 14].

For selecting training samples, we start by randomly picking points from the input parameter domain, and then execute the respective micro-benchmarks and observe Wayeb’s MCC scores. We call this initial set of configurations c , paired with MCC_c scores, D_{init} . Subsequently, using Bayesian inference, the first posteriors are calculated and the expected result is illustrated by comparing the prior in Figure 3a against the posterior in Figure 3b.

After D_{init} , the next micro-benchmarks are selected using an acquisition function $a(c)$. The acquisition function guides the selection of the next evaluation point by quantifying the utility of sampling a particular point x in the input space i.e., the domain of Wayeb’s configurations. $a(c)$ balances exploration and exploitation. Exploration involves sampling c configurations in the input space that are not yet well-explored or that have high uncertainty associated with them, while exploitation involves sampling c points that are likely to yield the best objective function values exploiting the current knowledge. For instance, in the plot of Figure 3c the acquisition function chooses the point in the input domain with the highest uncertainty. Different acquisition functions introduce stochasticity in the BO process by incorporating uncertainty estimates from the probabilistic model. BO concludes either when a micro-benchmark budget is depleted or when the value of $f(c)$ converges. Figure 3d illustrates a GPR with minimal uncertainty around its mean values, after the microbenchmark budget has been depleted.

Figure 4 illustrates the architecture of offCEf, comprising a Model Factory alongside a Controller. The Model Factory includes a Wayeb Server that utilises historical training and validation datasets to construct and evaluate PSTs. The Controller, includes the BO optimiser which is executed offline on a historical dataset. The Controller initialises BO by providing a set of configurations i.e., c vectors to the Model Factory, which, respectively, conducts the prescribed micro-benchmarks, saves temporarily the candidate PSTs, and sends reports to the Controller. The Controller will use these reports for updating the GPR surrogate model of BO.

offCEf deploys the PST that is expected to maximise MCC based on the hyperparameter vector c_{opt} calculated by BO. On the other hand, offCEf suffers from several disadvantages: (i) it drives its decisions by attributing equal importance to cumulative performance metric statistics, while in a streaming setup we often need to take into consideration only a sliding window of recent measurements and defy obsolete ones; (ii) it cannot optimise CEF hyperparameters at run-time which is a crucial limitation, since fluctuations in the input’s statistical properties in streaming settings is the norm rather than an infrequent situation; (iii) it cannot distinguish whether the hyperparameters for training PSTs should be adjusted through BO or if it is only the Wayeb’s PST that should be retrained, without changing hyperparameters. RTCEf, presented below, addresses these issues.

4.2 CEF Over Evolving Data Streams

We propose RTCEf, which is built with three major goals in mind. First, it updates at run-time PSTs according to input data evolutions; second, it performs CEF without disruptions, i.e., PST updating does not cause delays on CEF; and third, it does not overuse resources for producing new PSTs. The architecture of RTCEf consists of five main services, acting as Kafka producers and consumers, running synergistically to ensure uninterrupted CEF and dynamic PST retraining or hyperparameter optimisation. Figure 5 illustrates these services and the communication links between them. Synchronisation of the various services is denoted by dotted arrows in Figure 5. Below with details the processing of each service comprising our framework.

Observer. In order to determine whether the MCC score of Wayeb has deteriorated, the quality of its forecasts must be monitored. This task is handled by the Observer service (right of Figure 5) which consumes MCC scores from the ‘Reports’ topic and, produces ‘retrain’ or ‘optimise’ instructions as indicated in Algorithm 1. Essentially, a retrain instruction requests a new PST for Wayeb without changing training hyperparameters. An optimisation instruction, requests a new PST, produced through hyperparameter optimisation. Note

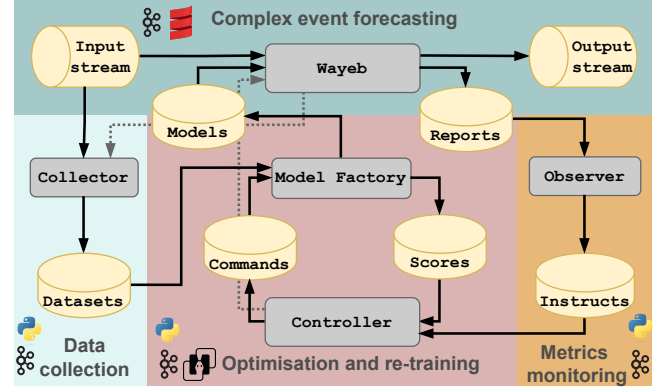


Figure 5: Architecture of RTCEf. Cylinders and rounded rectangles denote topics and services respectively. For simplicity, we omit synchronisation topics; instead we use gray arrows.

Algorithm 1 Observer service

Require: $k, guard_n, max_slope, min_score$

```

1:  $scores \leftarrow []$ 
2:  $guard \leftarrow -1$ 
3: while True do
4:    $score_i \leftarrow \text{consume}(\text{Reports})$ 
5:    $scores.\text{update}(score_i, k)$ 
6:    $pit\_cond \leftarrow score_i < min\_score$ 
7:    $slope\_cond \leftarrow \text{False}$ 
8:   if  $guard \geq 0$  then  $guard \leftarrow guard - 1$ 
9:   if  $\text{len}(|scores|) > 2$  then
10:     $(a_i, b_i) \leftarrow \text{fit\_trend}(scores)$ 
11:     $slope\_cond \leftarrow a_i < max\_slope$ 
12:   if  $(slope\_cond \text{ and } guard \geq 0)$  or  $pit\_cond$  then
13:     send("instructions", "optimise")
14:      $guard \leftarrow guard\_n$  ▷ New guard period
15:   else if  $slope\_cond$  then
16:     send("instructions", "retrain")
17:      $guard \leftarrow guard\_n$  ▷ New guard period

```

that hyperparameter optimisation will provide the best possible hyperparameters, but can be costly procedure, whereas retraining on an updated dataset is a cheaper process. We describe Algorithm 1 following its illustrative execution example for maritime situational awareness presented in Figure 6. The Observer continuously consumes MCC scores from Wayeb and retains the k most recent MCC scores to evaluate the performance trend. In the example of Figure 6, Wayeb begins with a PST, referred to as PST_{w_0} , created using configuration c_{w_0} . The Observer records the MCC Score at w_0 , however at this point no decision is made since fewer than $k = 3$ scores have been collected. Once the Observer has at least k scores, it computes the first degree polynomial $z_i(x) = a_i x + b_i$ (a trend line) so that a_i and b_i minimise

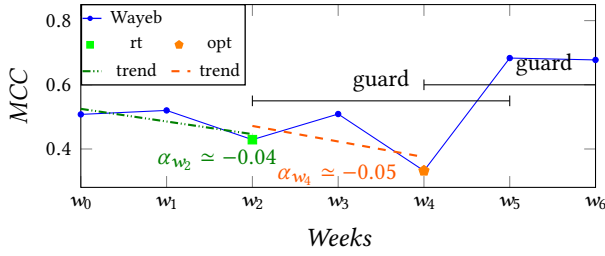


Figure 6: Execution example of the Observer. ‘opt’ and ‘rt’ stand for ‘optimisation’ and ‘retraining’ respectively. Dashed lines correspond to the trend lines associated with the Observer’s instructions at w_2 and w_4 , and black lines correspond to guard periods.

the squared error $E = \sum_{j=0}^{j=k} [z_i(x_j) - y_j]^2$ for $x_j = j$ and $y_j = score_{i-k+j}$, where i is an increasing integer denoting the ID of the current score (lines 9, 10). If the slope (a_i) of $z_i(x)$ is negative, indicating decrease in performance, and less than a $max_slope \in \mathbb{R}^-$ parameter (line 11) then a ‘retrain’ instruction is produced (lines 15-17). In the example, by week w_2 , the Observer has MCC scores for w_0, w_1, w_2 . Using these points, the Observer computes the trend line z_{w_2} with a slope $\alpha_{w_2} = -0.04$ which is steeper than $max_slope = -0.02$. To remedy this behaviour, the Observer issues a retrain instruction. As a result, a new PST, referred to as PST_{w_2} , is created using the same configuration as PST_{w_0} , i.e., c_{w_0} . This occurs because retraining updates the PST without modifying Wayeb’s hyperparameters.

Intuitively, forecasting performance deterioration, demonstrated by $a_i < max_slope$, shortly after a new PST deployment, indicates that the new PST failed and hyperparameter optimisation should thus be performed. To this end, we place each newly deployed PST in a guard period (lines 14,17). A guard period starts after a PST is deployed, and ends after $guard_n$ performance reports. If the performance of a PST under a guard period deteriorates ($a_i < max_slope$) then a hyperparameter optimisation instruction is produced (lines 12,13). If on the other hand, $a_i < max_slope$ is satisfied after $guard_n$ reports, then a ‘retrain’ instruction is produced for which a new “guard” period begins. In the example of Figure 6, a guard period begins at week w_2 and will last for $guard_n = 4$ reports, i.e., until w_5 . While PST_{w_2} shows improvement at w_3 , at w_4 performance drops again. The performance drop is also confirmed by the slope of -0.05 computed by the Observer using the MCC scores from w_2 to w_4 . Since the slope α_{w_4} is again below max_slope , but this time a guard period is active, the Observer issues a hyperparameter optimisation instruction instead of retraining. Consequently, a new PST_{w_4} is produced through hyperparameter optimisation, resulting in an updated configuration

c_{w_4} , and a new guard period starting at w_4 . Finally, to avoid pitfalls whereby the score drops suddenly very low, we employ an additional condition: if the score of a report is lower than a threshold min_score (line 6) then the Observer asks directly for ‘optimisation’ and omits a ‘retrain’ instruction.

Wayeb. The CEF part of RTCEF (top of Figure 5) contains Wayeb. In addition to reading timestamped simple events from the input stream and producing an output stream of CE forecasts, Wayeb produces a stream CEF forecasting performance reports, equally distanced by $reporting_distance$, and continuously monitors the ‘Models’ topic, which contains updated PSTs. When a new PST is made available in the Models topic, Wayeb replaces its PST with the latest available version. Recall that, to produce a CE forecast, Wayeb will utilise both the automaton corresponding to the symbolic regular expression defining a CE and the PST (see Section 2). The automaton retains information about the current state (q) and the next states that can lead to an accepting run, while the PST is used for producing the next symbol probabilities and therefore the waiting-time distribution for state q (W_q). Below, we show Wayeb PST update is “lossless” i.e., upon PST replacement, any run can continue from its current state and produce forecasts using the new PST.

Proposition 1. Given a stream $S = \{\sigma_0, \sigma_1, \dots, \sigma_k\}$ where σ_i are symbols, a SRE R , its corresponding automaton A_R , and a PST T with order $m \in [m_l, m_u]$, the replacement of T with a T' of order $m' \in [m_l, m_u]$ is lossless at any position i of the stream S if the last m_u symbols from the σ_i are available.

PROOF. We prove Proposition 1 by contradiction. Given S, R, A_R and T with order $m \in [m_l, m_u]$, assume that T is replaced with a PST T' with order m' at a position i . Now, assume that there exists a run that cannot continue from its current state q . This is not possible as the SRE R remains the same and therefore the automaton A_R is also the same, consequently the run can continue from q . Next, we assume that there exists a run with current state q for which the next symbol probabilities and the corresponding waiting-time distribution cannot be computed at position i under T' . Recall, that for an automaton run at position j and a current state q , the next symbol probability (and W_q) can be computed using the PST T and $S_{[j-m+1, j]}$, denoting the subset of S containing the symbols $\{\sigma_{j-m+1}, \dots, \sigma_j\}$. Since, $S_{[i-m_u+1, i]}$ is available, again the next symbol probabilities as well as the waiting-time distribution for any state q can be computed with T' as $S_{[i-m'+1, i]} \subseteq S_{[i-m_u+1, i]}$. \square

Proposition 1 states that updating a PST T with a new T' can be lossless if: first, the order m of any new PST lies within the same range $[m_l, m_u]$; and, second, the last m symbols up to the moment of the replacement are available. RTCEF ensures both conditions are satisfied. Consider, for example,

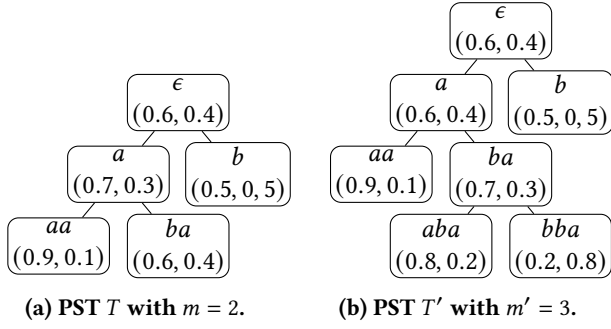


Figure 7: PST T with $m = 2$ and updated PST T' with $m = 3$, for $R := (speed > 10) \cdot (speed > 10)$ and the symbols $a = '(speed > 10)'$ and $b = '\neg(speed > 10)'$.

the regular expression $R := (speed > 10) \cdot (speed > 10)$, presented earlier in Section 2 as well as its corresponding automaton illustrated Figure 1. A PST T with order $m = 2$, along with a revised PST T' with order $m' = 3$, and $m' < m_u$, for R are illustrated in Figures 7a and 7b respectively. Here, ‘ a ’ corresponds to the symbol ‘ $(speed > 10)$ ’, while ‘ b ’ corresponds to the symbol ‘ $\neg(speed > 10)$ ’. Assume that CEF begins with PST T and processes an input stream $S = \{b, a, b, a\}$. After consuming S , the current state of the automaton of Figure 1 is 1, while the probability of completion in one step, i.e., receiving another a , is 0.6 (see the ba node in 7a). At this point, if we choose to replace T with T' , for a lossless transition we need at most the last 3 symbols from S —the order of T' is 3. Since these symbols are available and the current state is 1, the new probability of completion in one step after consuming S , and by considering T' , is 0.8 (see the aba node in Figure 7b). Notably, PST replacement, can be executed in linear time with respect to the number of automata runs and in practice happens in negligible time.

Collector. Training datasets evolve over time. Therefore, the data collection part of RTCEF (left of Figure 5) includes the Collector service, a data processing module organising and storing subsets of the input stream that may be used for retraining or hyperparameter optimisation. The Collector service consumes the input stream (see ‘Data collection’ in Figure 5), in parallel to Wayeb, and stores subsets of it in time buckets of fixed *bucket_size*. The Collector gathers data in a sliding window manner, emitting a new dataset version, containing *dt_size* buckets, to the ‘Datasets’ topic as soon as the last bucket in the range is full. Old buckets that no longer serve a purpose for training, are deleted for space economy.

Controller. The Controller service, based on the instructions of the Observer, initialises hyperparameter optimisation procedures, during which it also serves as the Bayesian optimiser, or retraining procedures, where it supplies Wayeb configurations. When optimisation is required, the Controller initiates the following three phases.

Initialisation phase: The Controller sets up the Bayesian optimiser. Similar to [10], we reuse micro-benchmarks from previous runs. Using the *retain_fraction* $\in [0, 1]$ parameter, we uniformly keep $\lfloor retain_fraction * all_samples \rfloor$ observations from the last BO run, where *all_samples* is the total number of micro-benchmarks. This speeds up optimisation while preserving useful information from previous runs.

Step phase: The Controller issues ‘train & test’ commands along with the hyperparameters suggested by the acquisition function—in our case the acquisition function is a combination of lower confidence bound, expected improvement, and probability of improvement. After each ‘train & test’ command, the Controller awaits the corresponding performance report i.e., the value of the $MCC(c)$ objective function. Upon receiving the performance report, the optimiser is updated with the new sample and the hyperparameters for the next step are suggested. The step phase ends when all micro-benchmarks are completed or if convergence is achieved.

Finalisation phase: Once optimisation concludes and the best hyperparameters are acquired, the Controller sends a finalisation message containing the ID of the best PST. Additionally, the Controller updates the previously best hyperparameters with the newly acquired ones, ensuring availability of the latter for subsequent ‘retrain’ instructions.

Model Factory. Similar to offCEF, the primary function of the Model Factory service is to train, test and send up-to-date PSTs to Wayeb. To do this, it will assemble and use the latest dataset version produced by the Collector. Upon receiving a ‘train’ command, the Model Factory trains a PST on the latest dataset and shares this new PST version with Wayeb.

For PST production through hyperparameter optimisation, upon receiving an ‘initialisation’ message, the Model Factory ‘locks’ the most recent assembled dataset so that the same dataset is used throughout the optimisation procedure. Next, during the ‘step’ phase, the Model Factory trains, saves and tests candidate PSTs on the locked dataset and reports MCC scores to the Controller. Finally, when the BO ‘finalisation’ message is received, the Model Factory sends the best performing PST to Wayeb. It is only at this point, that Wayeb will stop momentarily for PST replacement.

5 Experimental Evaluation

We evaluate our framework on two real-world use-cases. First, in maritime situational awareness, maritime CEs of interest are forecast over real vessel position streams. Second, in credit card fraud management, fraudulent activity is forecast over synthetic transaction data. We describe our experimental setup and then we present our findings.

5.1 Experimental Setup

5.1.1 Datasets & patterns. We present the datasets we employ and the patterns we use for forecasting CEs of interest.

Maritime situational awareness. We use a real-world, publicly available, maritime dataset containing 18M spatio-temporal positional AIS (Automatic Identification System) messages transmitted between October 1st 2016 and 31st March 2016 (6 months), from 5K vessels sailing in the Atlantic Ocean around the port of Brest, France [29]. AIS allows the transmission of information such as the current speed, heading and coordinates of vessels, as well as, ancillary static information such as destination and ship type. We evaluate RTCEF on a maritime pattern, which expresses the arrival of a vessel at the main port of Brest [32]. This pattern is derived after discussions with domain experts from a large, European maritime service provider [27, 28]:

$$R_{port} := (\neg InPort(Brest))^* \cdot (\neg InPort(Brest)) \cdot (\neg InPort(Brest)) \cdot (InPort(Brest)) \quad (2)$$

$InPort(Brest)$ is true when a vessel is within 5 km from the port of Brest. Recall that ‘ \neg ’, ‘ $*$ ’ and ‘ \cdot ’ correspond to negation, iteration (Kleene star) and sequence respectively (see Section 2). Consequently, R_{port} is satisfied if a sequence of at least three events occur. At least two require the vessel to be away from the port—thus limiting false positives from noisy entrances—, while the last denotes that the vessel has entered the port. This CE is important for port management and logistics reasons. We also perform experiments for a CE named R_{fish} defined as follows:

$$R_{fish} := (\neg InArea(Fishing))^* \cdot (\neg InArea(Fishing)) \cdot (\neg InArea(Fishing)) \cdot (InArea(Fishing) \wedge \neg SpeedRange(Fishing))^* \cdot (InArea(Fishing) \wedge SpeedRange(Fishing)) \quad (3)$$

$InArea(Fishing)$ is true when a vessel is within a fishing area, while $speedRange$ is a predicate satisfied when the vessel has fishing speed [28]. Therefore, R_{fish} is satisfied when initially a vessel is outside a fishing area, then the vessel enters the fishing area and; at some point while it is within the fishing area, it has fishing speed. Monitoring (illegal) fishing is important for environmental and sustainability reasons.

To cross validate our approach, we create 6 datasets MD_i , $i \in [0, 5]$ by shifting the starting month in a cyclic manner:

$$MD_i = \parallel_{j=0}^{j=5} month_{(j+i) \bmod 6}$$

where \parallel denotes the operation of concatenating two datasets, and $month_k$ corresponds to month k of the original dataset.

Credit card fraud management. We use a synthetic dataset provided by Feedzai² containing 1M credit card transaction events taking place over a period of 82 weeks. Each event contains, among others, the card ID, the amount and time of the transaction. We evaluate RTCEF on a pattern representing a fraudulent behaviour as a sequence of consecutive increasing transactions. Again, this pattern was determined

after discussions with domain experts from a large European credit card management service provider [3]:

$$R_{cards} := (amDiff > 0) \cdot (amDiff > 0) \cdot (amDiff > 0) \cdot (amDiff > 0) \cdot (amDiff > 0) \cdot (amDiff > 0) \quad (4)$$

We enrich events of the input stream with an additional attribute $amDiff$ which is equal to the difference between the previous transaction and the current one. Therefore, R_{cards} is satisfied when 8 consecutive transactions happen with increasing amounts. In order to simulate evolving fraud, we modify the financial dataset by changing randomly every 4 to 8 weeks the range of the highly correlated feature $amountDiff$. Similar to the maritime dataset, for validating our results, we create 21 datasets FD_i , $i \in [0, 20]$ by shifting 4 weeks the start of each dataset in a cyclic manner.

5.1.2 Initialisation. We perform offline hyperparameter optimisation with offCEF on the first four weeks of each dataset FD/MD_i and use the resulting PST, hyperparameters and micro-benchmark samples for initialising RTCEF. To showcase the benefit of RTCEF, we also perform CEF with static PSTs yielded by offCEF (see Section 4.1): i.e., for each FD_i/MD_i we adopt the stationarity assumption and perform CEF using the corresponding initial PST of each dataset. In what follows, the experiments that utilise the run-time adaptation framework are labelled with ‘RTCEF’ while experiments that are performed only with offline optimised static PSTs models are labelled with ‘offCEF’. Both offCEF and RTCEF ingest the input stream with the maximum speed of Kafka.

offCEF and RTCEF are implemented in Python 3.9.18, while the Kafka version was 3.5.2. Messages are formatted in JSON, and serialised/deserialised using Apache AVRO format. For BO, we use the scikit-optimize library 0.9.0. The experiments are conducted on a server running Debian 12 with an AMD EPYC 7543 32-Core Processor and 400G of RAM. Each service of RTCEF runs on its own dedicated core. Our framework is open-source and our experiments are reproducible.

5.2 Experimental Results

Figures 8 and 10 show MCC over time for R_{port} and R_{cards} (see Definitions (2) and (4) respectively), along with the score improvements when using RTCEF as opposed to offCEF for the maritime $MD_{0/2/3/5}$ and financial $FD_{0/1/8/17}$ datasets respectively. Results concerning MD_0 and FD_0 —the datasets in their original order—show that offCEF demonstrates, in both cases, poor performance and significant fluctuations in MCC scores over time. RTCEF, on the other hand, improves scores and reduces fluctuations in both cases.

Maritime situational awareness. For MD_0 —the maritime dataset in its original order—RTCEF dramatically improves

²<https://feedzai.com>

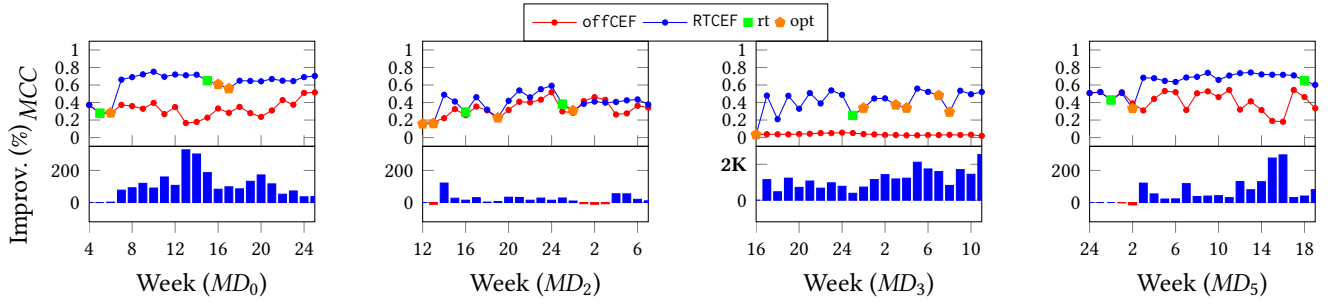


Figure 8: Experimental results for maritime datasets $MD_{0/2/3/5}$ with R_{port} . ‘rt’ and ‘opt’ stand for ‘retrain’ and ‘optimisation’ respectively. The plots show MCC (upper part) and MCC improvement (lower part) of ‘RTCEf’ relative to ‘offCEf’ over time.

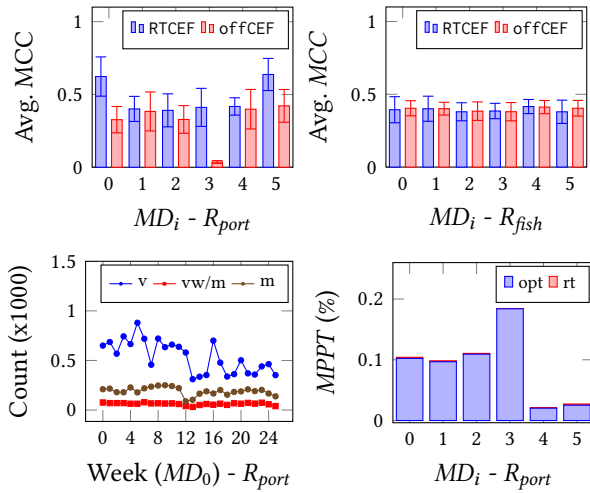


Figure 9: Cumulative results for the maritime situational awareness use-case. Avg MCC (top) per MD_i for R_{port} and R_{fish} . Dataset and CER characteristics (bottom left). ‘v’, ‘vw/m’ and ‘m’ stand for ‘vessels’, ‘vessels with matches’ and matches respectively. MPPT i.e., mean percentage of time spent every four weeks for production of PSTs per MD_i for R_{port} (bottom right).

MCC up to $\sim 300\%$ following retraining and optimisation procedures in weeks 5 and 6, respectively. A similar pattern is observed on the MD_5 dataset. On dataset MD_2 , although improvement is not as prominent as with $MD_{0/3/5}$, on average RTCEf improves significantly MCC (see Figure 9 top-left). On the MD_3 case, results show that the initial PST, generated by offCEf underperforms on weeks 16 to 19. However, this behaviour is immediately cured when the Observer requests optimisation in the first running week (see orange dot on week 16 of Figure 8 - MD_3)—this is due to the score being less than min_score (see Algorithm 1). We attribute the low scores of the initial PST of the MD_3 dataset on the lack of vessels passing through the monitoring area on that period (see

Figure 9 bottom-left). Figure 9 top-left, shows that the average MCC for each dataset MD_i ($i \in [0, 5]$) when using RTCEf is consistently higher than that achieved via a single PST trained only on the first four weeks of each dataset (offCEf). In Figure 9 (top-right) we report results concerning the R_{fish} CE (see Definition (3)). For the R_{fish} pattern there are no data evolutions in the input that affect CEF performance, therefore in this case, the results show that when data evolutions that affect PST performance at run-time are not present, the use of RTCEf does not affect forecasting performance.

Concerning processing efficiency, interruptions in CEF are minimal (see Proposition 1) as new PSTs are produced in parallel to CEF, thus efficiency and throughput of CEF remain unaffected. However, when a new PST request arises, new PST versions arrive with some delay. Recall, that until a new PST is available, Wayeb consumes the input stream, in parallel to the PST creation procedures, with the already deployed PST. Figure 9 (bottom-right) shows the mean percentage of time spent every four weeks for production of PSTs (we denote this value as MPPT) involving the R_{port} pattern. The results show that every four weeks, on average less than 0.2 % of time is spent for PST production (roughly 80 minutes in a period of four weeks) for all datasets MD_i . Consequently, RTCEf spends minimal time every four weeks for PST production, thus ensuring minimal delays and a resource-friendly behaviour.

Financial fraud management. Figure 10 shows that for FD_0 RTCEf overcomes input data evolutions, and significantly outperforms offCEf. A similar pattern is observed also for datasets FD_8 and FD_{17} (Figure 10), where changes in the input stream drop the MCC score of offCEf to almost 0. Again in this case, RTCEf, adapts and maintains overall a steady MCC over time. An interesting experiment is that of dataset FD_1 (Figure 10 - FD_1). Here, after hyperparameter optimisation is requested on week 26, RTCEf produces steadily an MCC score of ~ 0.75 . Conversely, offCEf on weeks 35-55 and 61-71 produces a score of ~ 0.85 , thus outperforming RTCEf. In this case, RTCEf fails to detect input data evolutions as after week

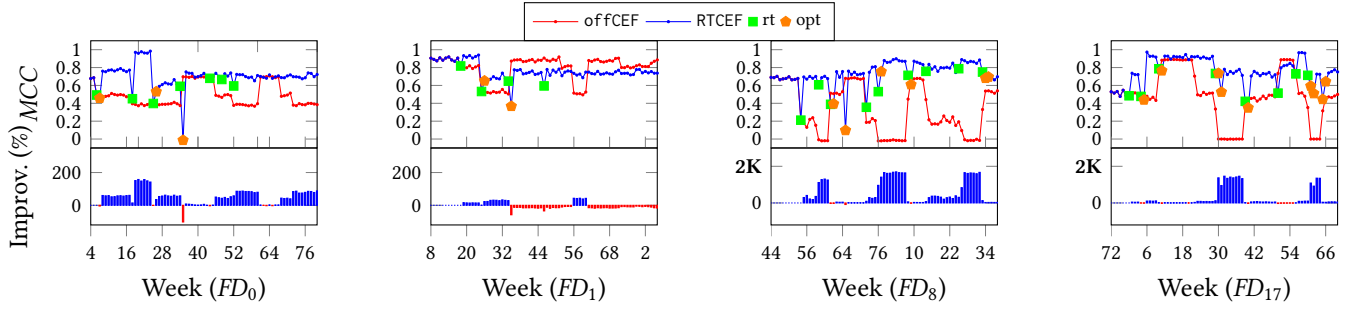


Figure 10: Experimental results for financial datasets $FD_{0/1/8/17}$ with R_{cards} . ‘rt’ and ‘opt’ stand for ‘retrain’ and ‘optimisation’ respectively. The plots show MCC (upper part) and MCC improvement (lower part) of ‘RTCEF’ relative to ‘offCEF’ over time.

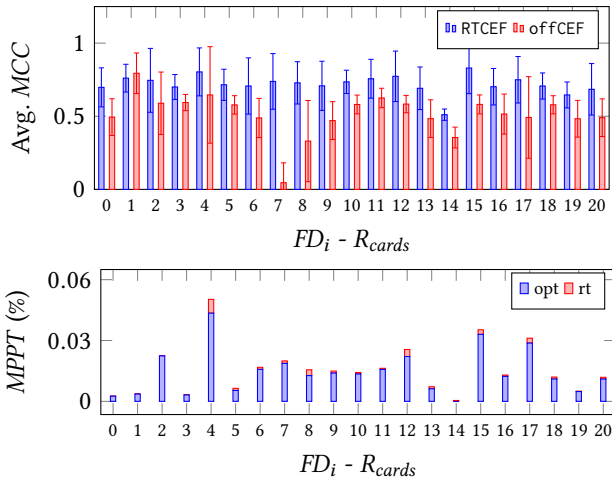


Figure 11: Cumulative results for the financial fraud management use-case i.e., avg MCC per FD_i for R_{cards} (top). MPPT i.e., mean percentage of time spent for PST production every four weeks per FD_i for R_{cards} (bottom).

26 there are no major fluctuations in MCC . Figure 11, shows that for FD_1 our framework has slightly less average MCC than the offline approach. However, for all other datasets FD_i , RTCEF significantly outperforms the offline approach (see Figure 11 top).

Similar to the experiments concerning maritime situational awareness, we report MPPT—i.e., the average time spent every four weeks for PST production (which takes place in parallel to CEF) due to retraining or optimisation—for all datasets FD_i and R_{cards} . Again, MPPT stays very low, with the maximum value being at most $\sim 0.05\%$. In other words, only 0.05% of time (~ 20 minutes) is spent for PST production every four weeks. Therefore, again RTCEF spends minimal time for PST production, thus ensuring an efficient resources use without sacrificing efficiency and throughput.

6 Related Work

Forecasting covers several areas such as time-series forecasting [24], general sequence prediction [5, 31], event sequence

prediction and point-of-interest recommendations [7, 21]. However, such methods focus on input event forecasting rather than CEF. Process mining, closely related to CEF [33], involves learning processes from activity logs and predicting process completions [13, 23]. Unlike CEF, which declaratively defines CEs, process mining focuses on transition systems and traces, typically consisting of long event sequences. Existing proposals in these areas often overlook CE patterns and primarily target input events or simple patterns, and cannot handle multiple variables of different types effectively. CEF aims to address such challenges, as outlined in various conceptual frameworks [8, 9, 15]. In our work specifically, we address these challenges by utilising Wayeb, a CEF engine that employs high-order Markov models [1, 2]. Furthermore, none of existing proposals (e.g., [20, 25, 26]) automate run-time adaptation without optimisation via exhaustive search, a gap addressed by our work, in addition to the fact that our approach optimises at run-time in a resource-friendly way.

The problem we address in this paper pertains to concept drift, i.e., evolutions in the data that invalidate the deployed model [16]. Our work is the first that tackles this problem specifically for CEF. For example, the work of Stavropoulos et al. [32] allows for offline CEF optimisation but does not allow run-time adaptation on dynamically evolving data streams, while concerning CEF optimisation itself, compared to our framework, it offers only a very restricted set of functionalities. EasyFlinkCEP [18], similar to RTCEF, uses BO to optimise the parallelism of FlinkCEP programs but lacks support for forecasting, focusing only on system-oriented metrics (e.g., throughput). Herodotou et al. [19] offer a comprehensive survey of machine learning-based techniques, including BO, for tuning the performance of Big Data management systems. Existing CER optimisation techniques focus on enhancing throughput i.e., the number of tuples processed per unit of time [12] while others focus on reducing processing latency and efficiently managing memory utilisation [12]. These approaches typically adapt traditional query optimisation techniques such as early predicate evaluation and query

rewriting to suit the context of CER. Giatrakos et al. [17] discuss techniques for executing parallel CER efficiently in geo-distributed settings. Notably, none of these approaches address run-time adaptation of CEF.

7 Summary

We presented, RTCEF, a novel framework for run-time adaptation of CEF. RTCEF involves several services running synergistically for uninterrupted run-time CEF and improved forecasting performance via lossless dynamic model updating. We evaluated our approach on two use-cases involving real-world and synthetic data and our experimental results show that there is a clear benefit using our framework as opposed to performing CEF with a single model in ‘offline’ fashion. We release publicly our framework in an open-source fashion.

Acknowledgments

This work was supported by the CREXDATA project, which received funding from the European Union’s Horizon Europe Programme, under grant agreement No 101092749.

References

- [1] Elias Alevizos, Alexander Artikis, and George Paliouras. 2018. Wayeb: a Tool for Complex Event Forecasting. In *LPAR*.
- [2] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. 2022. Complex event forecasting with prediction suffix trees. *VLDB J.* 31, 1 (2022), 157–180.
- [3] Alexander Artikis, Nikos Katzouris, Ivo Correia, Chris Baber, Natan Morar, Inna Skarbovsky, Fabiana Fournier, and Georgios Paliouras. 2017. A Prototype for Credit Card Fraud Management: Industry Paper. In *DEBS*. 249–260.
- [4] Alexander Artikis and Dimitris Zissis (Eds.). 2021. *Guide to Maritime Informatics*. Springer.
- [5] Ron Begleiter, Ran El-Yaniv, and Golan Yona. 2004. On Prediction Using Variable Order Markov Models. *J. Artif. Intell. Res.* 22 (2004), 385–421.
- [6] Eric Brochu, Vlad M. Cora, and Nando de Freitas. 2010. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning.
- [7] Buru Chang, Yonggyu Park, Donghyeon Park, Seongsoon Kim, and Jaewoo Kang. 2018. Content-Aware Hierarchical Point-of-Interest Embedding Model for Successive POI Recommendation. In *IJCAI*.
- [8] Maximilian Christ, Julian Krumeich, and Andreas W. Kempa-Liehr. 2016. Integrating Predictive Analytics into Complex Event Processing by Using Conditional Density Estimations. In *EDOC Workshops*.
- [9] Yagil Engel and Opher Etzion. 2011. Towards proactive event-driven computing. In *DEBS*.
- [10] Matthias Feurer, Jost Springenberg, and Frank Hutter. 2015. Initializing Bayesian Hyperparameter Optimization via Meta-Learning. *AAAI* 29 (2015).
- [11] Giannis Fikioris, Kostas Patroumpas, Alexander Artikis, Georgios Paliouras, and Manolis Pitsikalis. 2020. Fine-Tuned Compressed Representations of Vessel Trajectories. In *CIKM*.
- [12] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, Minos N. Garofalakis, Michael Kamp, and Michael Mock. 2017. Issues in complex event processing: Status and prospects in the Big Data era. *J. Syst. Softw.* 127 (2017), 217–236.
- [13] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, and Fredrik Milani. 2018. Predictive Process Monitoring Methods: Which One Suits Me Best?. In *BPM*.
- [14] Peter I. Frazier. 2018. A Tutorial on Bayesian Optimization.
- [15] Lajos Jeno Fülöp, Árpád Beszédes, Gabriella Toth, Hunor Demeter, László Vidács, and Lóránt Farkas. 2012. Predictive complex event processing: a conceptual framework for combining complex event processing and predictive analytics. In *BCI*.
- [16] João Gama, Indrune Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Comput. Surv.* 46 (2014), 189:1–189:38.
- [17] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Complex event recognition in the Big Data era: a survey. *VLDB J.* 29, 1 (2020), 313–352.
- [18] Nikos Giatrakos, Eleni Kougioumtzi, Antonios Kontaxakis, Antonios Deligiannakis, and Yannis Kotidis. 2021. EasyFlinkCEP: Big Event Data Analytics for Everyone. In *CIKM*.
- [19] Herodotos Herodotou, Yuxing Chen, and Jiaheng Lu. 2020. A Survey on Automatic Parameter Tuning for Big Data Processing Systems. *ACM Comput. Surv.* 53, 2 (2020), 43:1–43:37.
- [20] Yan Li, Tingjian Ge, and Cindy X. Chen. 2020. Data Stream Event Prediction Based on Timing Knowledge and State Transitions. *Proc. VLDB Endow.* 13, 10 (2020), 1779–1792.
- [21] Zhongyang Li, Xiao Ding, and Ting Liu. 2018. Constructing Narrative Event Evolutionary Graph for Script Event Prediction. In *IJCAI*.
- [22] Alessandro Margara and Gianpaolo Cugola. 2011. Processing flows of information: from data stream to complex event processing. In *DEBS*.
- [23] Alfonso Eduardo Márquez-Chamorro, Manuel Resinas, and Antonio Ruiz-Cortés. 2018. Predictive Monitoring of Business Processes: A Survey. *IEEE Trans. Services Computing* 11, 6 (2018), 962–977.
- [24] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. 2015. *Introduction to time series analysis and forecasting*. John Wiley & Sons.
- [25] Vinod Muthusamy, Haifeng Liu, and Hans-Arno Jacobsen. 2010. Predictive publish/subscribe matching. In *DEBS*.
- [26] Suraj Pandey, Surya Nepal, and Shiping Chen. 2011. A test-bed for the evaluation of business process prediction techniques. In *Collaborate-Com*.
- [27] Kostas Patroumpas, Alexander Artikis, Nikos Katzouris, Marios Voda, Yannis Theodoridis, and Nikos Pelekis. 2015. Event Recognition for Maritime Surveillance. In *EDBT*.
- [28] Manolis Pitsikalis, Alexander Artikis, Richard Dreó, Cyril Ray, Elena Camossi, and Anne-Laure Jousselme. 2019. Composite Event Recognition for Maritime Monitoring. In *DEBS*.
- [29] Cyril Ray, Richard Dreó, Elena Camossi, Anne-Laure Jousselme, and Clément Iphar. 2019. Heterogeneous integrated dataset for Maritime Intelligence, surveillance, and reconnaissance. *Data in Brief* 25 (2019), 104141.
- [30] Dana Ron, Yoram Singer, and Naftali Tishby. 1993. The Power of Amnesia. In *NIPS*.
- [31] Dana Ron, Yoram Singer, and Naftali Tishby. 1996. The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. *Machine Learning* 25, 2-3 (1996), 117–149.
- [32] Vasileios Stavropoulos, Elias Alevizos, Nikos Giatrakos, and Alexander Artikis. 2022. Optimizing complex event forecasting. In *DEBS*.
- [33] Wil Van Der Aalst. 2011. *Process mining: discovery, conformance and enhancement of business processes*.
- [34] Li Yang and Abdallah Shami. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415 (2020), 295–316.