Towards Run-Time Adaptation of Complex Event Forecasting

Manolis Pitsikalis[1,*], Elias Alevizos[1,2], Nikos Giatrakos[3] and Alexander Artikis[4,1]

[1]NCSR Demokritos, Athens, Greece
[2]The American College of Greece, Athens, Greece
[3]Technical University of Crete, Chania, Greece
[4]University of Piraeus, Greece

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, maritime vessels adapt their routes based on weather or human-caused factors. 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 a real-world maritime use-case and our reproducible results show that our proposed approach has significant benefits in terms of forecasting performance without sacrificing efficiency.

Keywords

complex event forecasting, run-time adaptation, distributed architecture

1. Introduction

Complex Event Forecasting (CEF) is akin to Complex Event Recognition (CER) [1, 2], 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 maritime situational awareness [3], the stream of simple events would contain positional messages of vessels, while the output stream would contain maritime CEs such as 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 [4, 5] whereby CEs such as (illegal) fishing or vessel rendezvous are detected or forecast over a stream of maritime data; credit card fraud management [5, 6] whereby frauds are detected or forecast over a stream of transaction data; and so on.

CEF operates over constantly evolving conditions. Moreover, CEF systems rely on probabilistic models trained on historical data [7, 8, 9]. This renders CEF systems inherently susceptible to evolutions in the input that can invalidate their underlying models. Additionally, as with the majority of trainable models, CEF models have hyperparameters that require fine tuning for optimal performance. Wayeb [5], 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 ongoing 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. We evaluate RTCEF on maritime situational awareness using real-world maritime data, and our results demonstrate that RTCEF has significant benefits for CEF over constantly evolving conditions.

The remainder of this paper is organised as follows. In Section 2 we present the necessary background. Next, in Section 3 we introduce `ofCEF`and RTCEF, while in Section 4 we present our experimental setting, and analyse our results. In Section 5 we mention works related to ours. Finally, in Section 6, we summarise and discuss future directions.

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 [5]. 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, a CEF engine introduced in [5], 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

Published in the Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference (March 25-28, 2025), Barcelona, Spain
*Corresponding author.
manospits@iit.demokritos.gr (M. Pitsikalis); alevizos.elias@iit.demokritos.gr (E. Alevizos); ngiatrakos@tuc.gr (N. Giatrakos); a.artikis@iit.demokritos.gr (A. Artikis)
https://manospits.github.io/ (M. Pitsikalis); http://users.softnet.tuc.gr/~ngiatrakos/ (N. Giatrakos); https://users.iit.demokritos.gr/~a.artikis/ (A. Artikis)
0000-0003-2959-2022 (M. Pitsikalis); 0000-0002-9260-0024 (E. Alevizos); 0000-0002-8218-707X (N. Giatrakos); 0000-0001-6899-4599 (A. Artikis)

Table 1
An example stream 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	...

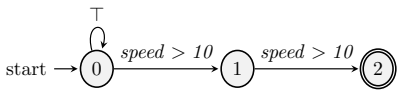


Figure 1: Streaming symbolic automaton created from the expression $R := (speed > 10) \cdot (speed > 10)$.

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 [1]. 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 [5]. 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 [5]. 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 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 $timestamp = 5$.

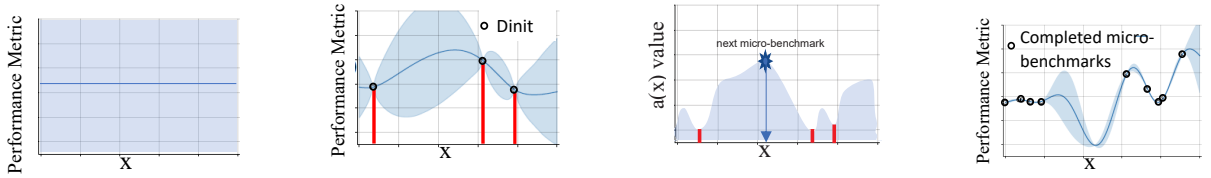
To perform CEF, Wayeb needs a probabilistic description for a symbolic automaton derived from a SRE. For this purpose, Wayeb employs Prediction Suffix Trees (PSTs) [10, 11]– a form of Variable-order Markov Models. Variable-order Markov Models, compared to fixed-order Markov models, capture longer-term 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 3 (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 [11, 6, 5]. 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 3 (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. Since 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), which is defined as follows:

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

where $NPV = \frac{TN}{TN+FN}$, $\textit{Specificity} = \frac{TN}{TN+FP}$, $FDR = 1 - \textit{Precision}$, $FNR = 1 - \textit{Recall}$, $FPR = 1 - \textit{Specificity}$ 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, where 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



(a) Before D_{init}, shaded area shows uncertainty of performance ($f(x)$) values.

(b) Fitting GPR around micro-benchmarks in D_{init} (red lines). Blue line shows expected performance.

(c) The acquisition function chooses next micro-benchmark x of high uncertainty.

(d) A GPR concludes after a number of BO micro-benchmarks.

Figure 2: Bayesian Optimisation Operation.

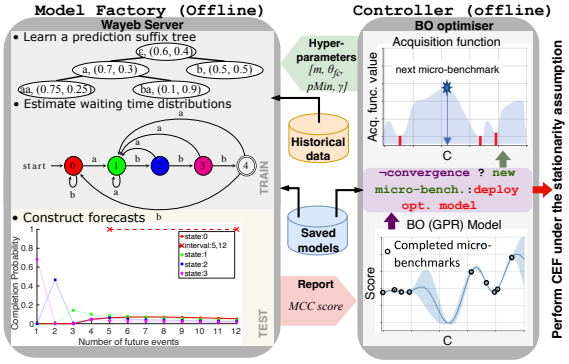


Figure 3: Architecture of the offline CEF optimiser (offCEF).

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. Run-Time CEF Adaptation

We first present 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.

3.1. 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:

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

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) [12, 13] 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 [14] BO does not require a high number of micro-benchmarks or an analytical formula [12, 13, 6]. 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:

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

$\mu_n(c)$ and $k_n(c, c')$ being the posterior mean and covariance functions updated through Bayesian inference [12, 13].

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 2a against the posterior in Figure 2b.

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 2c 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 2d illustrates a GPR with minimal uncertainty around its mean values, after the microbenchmark budget has been depleted.

Figure 3 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 combination value 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.

3.2. 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 undisrupted CEF and dynamic PST retraining or hyperparameter optimisation. Figure 4 illustrates these services and the communication links between

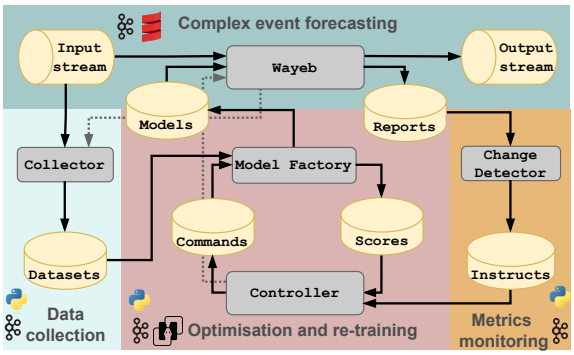


Figure 4: Architecture of RTCEF. Cylinders and rounded rectangles denote topics and services respectively. For simplicity, we omit synchronisation topics; instead we use gray arrows.

them. Synchronisation of the various services is denoted by dotted arrows in Figure 4. Below we describe in detail the services comprising our framework.

Change Detector. 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 Change Detector service (right of Figure 4) 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 that while hyperparameter optimisation will provide the best possible hyperparameters, it can be a costly procedure. On the other hand, retraining on an updated dataset is a cheaper process.

The decision process of Algorithm 1 is summarised in Figure 5, while an illustrative execution of Algorithm 1 for maritime situational awareness is presented in Figure 6. We describe Algorithm 1 following the example execution of Figure 6. The Change Detector 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 Change Detector 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 Change Detector 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 the following squared error:

$$E = \sum_{j=0}^{j=k} \left[z_i(x_j) - y_j \right]^2$$

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 Change Detector has MCC scores for w_0, w_1, w_2. Using these points, the Change Detector 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 Change Detector 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

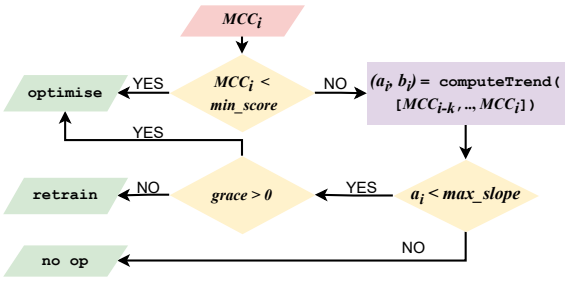


Figure 5: Decision process of the Change Detector. Pink and green parallelograms correspond to input and output information respectively, rhombi correspond to conditionals, and rectangles correspond to function calls.

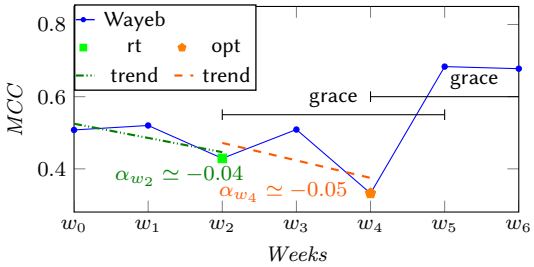


Figure 6: Execution sample of the Change Detector for maritime situational awareness. 'opt' and 'rt' stand for 'optimisation' and 'retraining' respectively. Dashed dotted, and dashed lines correspond to the trend lines associated with the retrain and optimisation instructions at t_2 and t_4 respectively, while black lines correspond to grace periods initiated at the issuance of retrain or optimisation instructions.

modifying Wayeb's hyperparameters.

Intuitively, forecasting performance deterioration i.e., $a_i < max\_slope$, shortly after a new PST deployment, indicates that the new PST failed and hyperparameter optimisation should thus be performed. We place each newly deployed PST in a grace period (lines 14,17). A grace period starts after a PST is deployed, and ends after $grace\_n$ performance reports. If the performance of a PST under a grace 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 $grace\_n$ reports, then a 'retrain' instruction is produced for which a new "grace" period begins. In the example of

Algorithm 1 Change Detector service

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

Figure 5 a grace period begins at week w_2 and will last for $grace\_n = 4$ 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 Change Detector 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 grace period is active, the Change Detector 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 grace 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 Change Detector asks directly for 'optimisation' and omits a 'retrain' instruction.

Wayeb. The CEF part of RTCEF (top of Figure 4) 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 of CEF forecasting performance reports 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. The automaton retains information about the current state (q) as well as 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). Notably, the process of replacing the PST with a new version can be executed in linear time with respect to the number of runs and in practice happens in negligible time.

Collector. Training datasets evolve over time. Therefore, the data collection part of RTCEF (left of Figure 4) 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. Therefore, the Collector service consumes the input stream (see 'Data collection' in Figure 4), in parallel to Wayeb, and stores subsets of it in time buckets. The Collector gathers data in a sliding window manner, emitting a new dataset version 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 Change Detector, 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 [15], we leverage micro-benchmarks from previous runs. Using the $retain\_percentage \in [0, 1]$ parameter, we uniformly keep $\lfloor retain\_percentage * all\_samples \rfloor$ observations from the last executed BO run, where $all\_samples$ is the total number of micro-

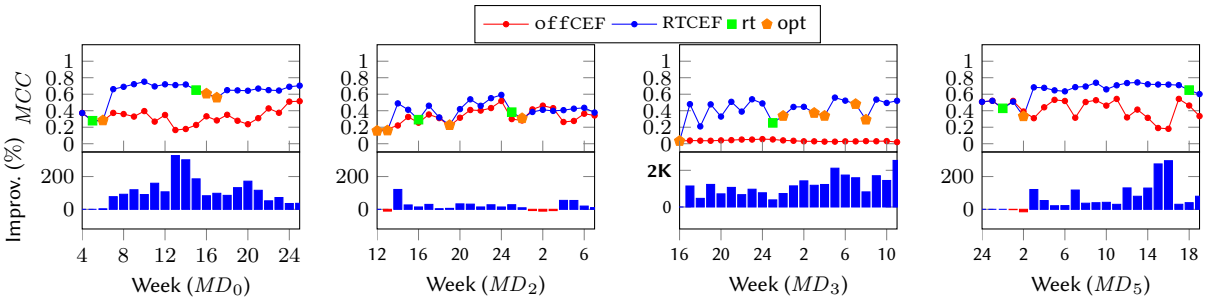


Figure 7: Experimental results for datasets $MD_{0/2/3/5}$ with R_{port}. 'rt' and 'opt' stand for 'retrain' and 'optimisation' respectively. The plots show MCC (upper part) and MCC improvement (lower part) of 'RTCEF' relative to 'offCEF' over time.

benchmarks. This allows us to accelerate optimisation, and retain 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 $f(c)$ 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 PST 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, including the ID of the best performing PST, is received, the Model Factory sends the best PST to Wayeb. It is only at this point, that Wayeb will stop momentarily for PST replacement.

4. Experimental Evaluation

We evaluate our framework on maritime situational awareness, where maritime CEs of interest are forecast over real vessel position streams.

4.1. Experimental Setup

We use a real-world, publicly available, maritime dataset containing 18M spatio-temporal positional AIS (Automatic

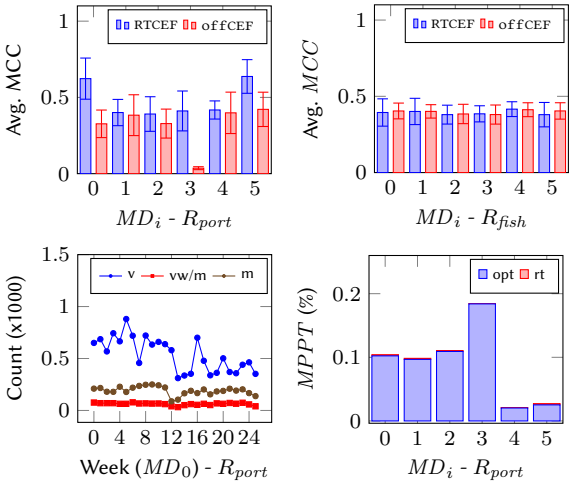


Figure 8: Avg MCC (top) per MD_i for R_{port} and R_{fish}. Dataset and CER characteristics (bottom left). 'v', 'vw/m' and 'm' stand for 'vessels', 'vessels with matches' and 'matches' respectively. MPPT i.e., mean percentage of time spent every four weeks for production of models per MD_i for R_{port} (bottom right).

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 [16]. 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 [6]. This pattern is derived after discussions with domain experts from a large, European maritime service provider [4, 17]:

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

$InPort(Brest)$ 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 [4]. 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:

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

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

We perform offline hyperparameter optimisation with offCEF on the first four weeks of each dataset MD_i and use the resulting model, hyperparameters and micro-benchmark samples for initialisation. To showcase, the benefit of RTCEF, we additionally perform CEF with static models yielded by offCEF (see Section 3.1): i.e., for each MD_i we adopt the stationarity assumption and perform CEF using the corresponding initial model of each dataset. In what follows, the experiments that utilise the run-time optimisation framework are labelled with 'RTCEF' while experiments that are performed only with offline optimised static models are labelled with 'offCEF'.

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[1].

4.2. Experimental Results

Figure 7 shows the evolution of MCC over time for R_{port} (see Definition (2)), along with the score improvements when using RTCEF as opposed to offCEF for the $MD_{0/2/3/5}$ datasets respectively. For MD_0—the dataset in its original order—RTCEF drammatically improves MCC up to $\sim 300\%$ following retraining and hyperparameter 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 MCC (see Figure 8 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 Change Detector requests hyperparameter optimisation in the first running week (see orange dot on week 16 of Figure 7 - MD_3)—this is due to the score being less than min_score (see Algorithm 1). We attribute the low scores of the initial model of the MD_3 dataset on the lack of vessels passing through the monitoring area on that period (see Figure 8 bottom-left). Figure 8 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 model trained only on the first four weeks of each dataset (offCEF). In Figure 8 (top-right) we report results concerning the R_{fish} CE. For the R_{fish} pattern (see Definition (3)) 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 model performance at run-time are not present, the use of RTCEF does not affect forecasting performance.

Concerning processing efficiency, interruptions in CEF are minimal as retraining or optimisation procedures take place 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 update procedures, with the already deployed PST. Figure 8 (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 as optimisation or retraining procedures are not overperformed.

5. Related Work

The problem addressed in this paper pertains to concept drift, i.e., evolutions in the data that invalidate the deployed model [18]. Our work is the first that tackles this problem specifically for CEF. For example, the work of Stavropoulos et al. [6] 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 [19], 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. [20] 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 [21] while others focus on reducing processing latency and efficiently managing memory utilisation [21]. 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. [1] discuss techniques for executing parallel CER efficiently in geo-distributed settings. Notably, none of the above address run-time CEF adaptation.

Forecasting covers several areas such as time-series forecasting [22], general sequence prediction [23, 10], event sequence prediction and point-of-interest recommendations [24, 25]. However, such methods primarily focus on input event forecasting rather than CEF. Process mining, closely related to CEF [26], involves learning processes from activity logs and predicting process completions [27, 28].

[1]Execution scripts in https://github.com/manospits/rtcef/tree/main/scripts

However, process mining often overlooks CE patterns. CEF aims to address such challenges, as outlined in various conceptual frameworks [29, 30, 31]. In our work specifically, we address these challenges by utilising Wayeb, a CEF engine that employs high-order Markov models [5, 9]. Furthermore, none of existing proposals (e.g., [7, 8, 32]) automate run-time adaptation in resource-friendly way.

6. Summary and Further Work

We presented, RTCEF, a novel framework for run-time optimisation of CEF. RTCEF involves several services running synergistically for undisrupted run-time CEF and improved performance via lossless dynamic model updating. We evaluated our approach on maritime situational awareness usecase involving real-world 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.

For future work, we plan to investigate additional data collection, and retrain vs optimisation policies. Furthermore, we aim to integrate parallel BO. Finally, we want to evaluate our framework on additional problems such as run-time CER query optimisation.

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] N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis, M. N. Garofalakis, Complex event recognition in the big data era: a survey, VLDB J. 29 (2020) 313–352.
[2] A. Margara, G. Cugola, Processing flows of information: from data stream to complex event processing, in: DEBS, 2011.
[3] A. Artikis, D. Zissis (Eds.), Guide to Maritime Informatics, Springer, 2021.
[4] M. Pitsikalis, A. Artikis, R. Dreo, C. Ray, E. Camossi, A.-L. Jousselme, Composite event recognition for maritime monitoring, in: DEBS, 2019.
[5] E. Alevizos, A. Artikis, G. Paliouras, Complex event forecasting with prediction suffix trees, VLDB J. 31 (2022) 157–180.
[6] V. Stavropoulos, E. Alevizos, N. Giatrakos, A. Artikis, Optimizing complex event forecasting, in: DEBS, 2022.
[7] S. Pandey, S. Nepal, S. Chen, A test-bed for the evaluation of business process prediction techniques, in: CollaborateCom, 2011.
[8] V. Muthusamy, H. Liu, H. Jacobsen, Predictive publish/subscribe matching, in: DEBS, 2010.
[9] E. Alevizos, A. Artikis, G. Paliouras, Wayeb: a tool for complex event forecasting, in: LPAR, 2018.
[10] D. Ron, Y. Singer, N. Tishby, The power of amnesia: Learning probabilistic automata with variable memory length, Machine Learning 25 (1996) 117–149.
[11] D. Ron, Y. Singer, N. Tishby, The power of amnesia, in: NIPS, 1993.
[12] E. Brochu, V. M. Cora, N. de Freitas, A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010.
[13] P. I. Frazier, A tutorial on bayesian optimization, 2018.
[14] L. Yang, A. Shami, On hyperparameter optimization of machine learning algorithms: Theory and practice, Neurocomputing 415 (2020) 295–316.
[15] M. Feurer, J. Springenberg, F. Hutter, Initializing bayesian hyperparameter optimization via meta-learning, AAAI 29 (2015).
[16] C. Ray, R. Dréo, E. Camossi, A.-L. Jousselme, C. Iphar, Heterogeneous integrated dataset for maritime intelligence, surveillance, and reconnaissance, Data in Brief 25 (2019) 104141.
[17] K. Patroumpas, A. Artikis, N. Katzouris, M. Vodas, Y. Theodoridis, N. Pelekis, Event recognition for maritime surveillance, in: EDBT, 2015.
[18] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (2014) 189:1–189:38.
[19] N. Giatrakos, E. Kougioumtzi, A. Kontaxakis, A. Deligiannakis, Y. Kotidis, Easyflinkcep: Big event data analytics for everyone, in: CIKM, 2021.
[20] H. Herodotou, Y. Chen, J. Lu, A survey on automatic parameter tuning for big data processing systems, ACM Comput. Surv. 53 (2020) 43:1–43:37.
[21] I. Flouris, N. Giatrakos, A. Deligiannakis, M. N. Garofalakis, M. Kamp, M. Mock, Issues in complex event processing: Status and prospects in the big data era, J. Syst. Softw. 127 (2017) 217–236.
[22] D. C. Montgomery, C. L. Jennings, M. Kulahci, Introduction to time series analysis and forecasting, John Wiley & Sons, 2015.
[23] R. Begleiter, R. El-Yaniv, G. Yona, On prediction using variable order markov models, J. Artif. Intell. Res. 22 (2004) 385–421.
[24] Z. Li, X. Ding, T. Liu, Constructing narrative event evolutionary graph for script event prediction, in: IJCAI, 2018.
[25] B. Chang, Y. Park, D. Park, S. Kim, J. Kang, Content-aware hierarchical point-of-interest embedding model for successive POI recommendation, in: IJCAI, 2018.
[26] W. Van Der Aalst, Process mining: discovery, conformance and enhancement of business processes, 2011.
[27] A. E. Márquez-Chamorro, M. Resinas, A. Ruiz-Cortés, Predictive monitoring of business processes: A survey, IEEE Trans. Services Computing 11 (2018) 962–977.
[28] C. D. Francescomarino, C. Ghidini, F. M. Maggi, F. Milani, Predictive process monitoring methods: Which one suits me best?, in: BPM, 2018.
[29] L. J. Fülöp, Á. Beszédes, G. Toth, H. Demeter, L. Vidács, L. Farkas, Predictive complex event processing: a conceptual framework for combining complex event processing and predictive analytics, in: BCI, 2012.
[30] Y. Engel, O. Etzion, Towards proactive event-driven computing, in: DEBS, 2011.
[31] M. Christ, J. Krumeich, A. W. Kempa-Liehr, Integrating predictive analytics into complex event processing by using conditional density estimations, in: EDOC Workshops, 2016.
[32] Y. Li, T. Ge, C. X. Chen, Data stream event prediction based on timing knowledge and state transitions, Proc. VLDB Endow. 13 (2020) 1779–1792.