

# Continuous Management of Machine Learning-Based Application Behavior

Marco Anisetti *Senior Member, IEEE*, Claudio A. Ardagna *Senior Member, IEEE*, Nicola Bena *Student Member, IEEE*, Ernesto Damiani *Senior Member, IEEE*, Paolo G. Panero

**Abstract**—Modern applications are increasingly driven by Machine Learning (ML) models whose non-deterministic behavior is affecting the entire application life cycle from design to operation. The pervasive adoption of ML is urgently calling for approaches that guarantee a stable non-functional behavior of ML-based applications over time and across model changes. To this aim, non-functional properties of ML models, such as privacy, confidentiality, fairness, and explainability, must be monitored, verified, and maintained. Existing approaches mostly focus on *i)* implementing solutions for classifier selection according to the functional behavior of ML models, *ii)* finding new algorithmic solutions, such as continuous re-training. In this paper, we propose a multi-model approach that aims to guarantee a stable non-functional behavior of ML-based applications. An architectural and methodological approach is provided to compare multiple ML models showing similar non-functional properties and select the model supporting stable non-functional behavior over time according to (dynamic and unpredictable) contextual changes. Our approach goes beyond the state of the art by providing a solution that continuously guarantees a stable non-functional behavior of ML-based applications, is ML algorithm-agnostic, and is driven by non-functional properties assessed on the ML models themselves. It consists of a two-step process working during application operation, where *model assessment* verifies non-functional properties of ML models trained and selected at development time, and *model substitution* guarantees continuous and stable support of non-functional properties. We experimentally evaluate our solution in a real-world scenario focusing on non-functional property fairness.

**Index Terms**—Assurance, Machine Learning, Multi-Armed Bandit, Non-Functional Properties



## 1 INTRODUCTION

Machine Learning (ML) has become the technique of choice to provide advanced functionalities and carry out tasks hardly achievable by traditional control and optimization algorithms [1]. Even the behavior, orchestration, and deployment parameters of distributed systems and services, possibly offered on the cloud-edge continuum, are increasingly based on ML models [2]. Concerns about the black-box nature of ML have led to a societal push that involves all components of society (policymakers, regulators, academic and industrial stakeholders, citizens) towards trustworthy and transparent ML, giving rise to legislative initiatives on artificial intelligence (e.g., the AI Act in Europe [3]).

This scenario introduces the need for solutions that continuously guarantee a stable non-functional behavior of ML-based applications, a task that is significantly more complex than mere QoS-based selection and composition (e.g., [4], [5], [6]). The focus of such a task is to assess the non-functional properties of ML models, such as privacy, confidentiality, fairness, and explainability, over time and across changes. The non-functional assessment of ML-based applications behavior has to cope with the ML models' complexity, low transparency, and continuous evolution [7], [8]. ML models in fact are affected by model and data

drifts, quality degradation, and accuracy loss, which may substantially impact on the quality and soundness of the application itself.

Recent research points to solutions where ML models evolve according to contextual changes (e.g., a shift in the incoming data distribution), typically via continuous re-training and peculiar training algorithms and ML models [9], [10], [11]. Other solutions consider *classifier selection* where a (set of) ML model is statically or dynamically selected according to some criteria [12], [13], [14], [15]; in this context, dynamic selection identifies the most suitable ML model for each data point at inference time. Ensembles have been also considered to increase ML robustness [16], [17], [18], [19], [20]. Finally, some solutions have initially discussed certification-based assessment of ML-based applications [7], [8], [21]. Current approaches however fall short in supporting the requirements of modern ML-based applications. On the one hand, they disregard stable application behavior and non-functional properties, which are increasingly mandated by law, in favor of accuracy maximization. On the other hand, they do not provide a general solution that applies to *any* non-functional properties and ML algorithms, and rather focus on specific, though relevant, properties (e.g., fairness) and algorithms (e.g., decision trees).

This paper fills in the above gaps by defining a multi-model approach that guarantees a stable non-functional behavior of ML-based applications. Similarly to dynamic classifier selection, our approach keeps a pool of ML models and one ML model at time is dynamically selected during inference according to a (set of) non-functional property; the selected ML model is replaced only when its non-functional

- M. Anisetti, C. A. Ardagna, N. Bena, E. Damiani, Paolo G. Panero are with the Department of Computer Science, Università degli Studi di Milano, Milano, Italy. E. Damiani is also with Khalifa University, Abu Dhabi, UAE.  
E-mail: {firstname.lastname}@unimi.it, paolo.panero@studenti.unimi.it

property degrades. Our approach is particularly suited for constrained and critical scenarios with (dynamic and unpredictable) contextual changes. In such scenarios online re-training and dynamic classifier/ensemble selection approaches *i)* have a larger overhead due to the expensive training and the need to select a model *for each data point*, and *ii)* can lead to unexpected application behavior due to the arrival of new, unpredictable input data.

Our multi-model approach is built on a two-step process working during application operation as follows. The first step, *model assessment*, verifies non-functional properties of ML models already trained and selected at development time. To this aim, we extend our previous work on Multi-Armed Bandit (MAB) [22], towards a dynamic MAB that assesses non-functional properties of ML models at run time. The second step, *model substitution*, is driven by the properties assessed at step *i)*, and guarantees a stable support for non-functional properties over time and across changes. Our approach can be used both as a complete solution for application behavior management (from design to operation), or to complement existing ML-based applications with a multi-model substitution approach.

Our contribution is threefold. We first propose a new definition of non-functional property of ML models. Our definition departs from traditional, attribute-based properties available in the literature (e.g., [23], [24]), and includes a scoring function at the basis of ML models comparison and selection. We extend the scope of traditional properties, which are mostly based on accuracy [12] or metrics unrelated to the model itself (e.g., the battery level of the device or latency [25]), to include non-functional properties such as fairness and integrity, often mandated by law. Though important, these properties are often neglected in literature [26]. We then describe our multi-model approach for managing the non-functional behavior of ML-based applications. Our approach defines a dynamic MAB for the assessment of the non-functional properties of ML models, and proposes two model substitution strategies built on it. The two strategies support the dynamic choice of the model with the best set of non-functional properties at run time, by ranking and substituting the models in a dynamically sized evaluation window, and performing additional early substitutions upon severe non-functional degradation using an assurance-based evaluation. We note that, although a plethora of assurance techniques exist for the verification of non-functional properties in traditional service-based applications [23], [27], [28], [29], the definition of rigorous assurance-based processes for ML-based applications is still more an art than a science [7], [8]. We finally extensively evaluate our solution focusing on non-functional property *fairness*.

The remainder of this paper is organized as follows. Section 2 presents our reference scenario and our approach at a glance. Section 3 describes our building blocks, including Static MAB that is later extended in Section 4 towards Dynamic MAB for non-functional ML model assessment. Section 5 presents the two strategies for model substitution. Section 6 describes our approach in an end-to-end walk-through. Section 7 presents an extensive experimental evaluation in a real scenario. Section 8 comparatively discusses the related work and Section 9 draws our conclusions.

## 2 OUR APPROACH AT A GLANCE

We consider a scenario where a service provider is willing to deploy an application (service workflow) whose behavior depends on an ML model. The service provider needs to maintain stable performance across time in terms of quality (e.g., accuracy of the model) and non-functional posture (e.g., fairness). Let us assume a scenario where the model behavior changes, such as model drift (e.g., due to online partial or full re-training) or data drift (e.g., service re-deployment or migration in the cloud-edge continuum), which are induced by modifications in the application operational conditions. To cope with this scenario, the service provider adopts a multi-model approach by designing and deploying multiple models that can be alternatively used depending on the context. This multi-model deployment can impact single or multiple nodes in cloud or cloud-edge scenarios. We note that the model behavior is evaluated at design time and continuously monitored at run time to decide which model to use during application operation. We also note that the service provider can decide to substitute or not the model in operation due to restrictions in the application environment, but is always capable of comparing the behavior of the model in use with the other alternative models and use this evidence to fine-tune them offline.

Figure 1 shows an overview of the above scenario and how we apply our multi-model approach to address the continuous management of ML-based application non-functional behavior.

Our approach starts at development time with a set of pre-trained, candidate ML models and statically selects the model with the best (set of) non-functional property to be used by the application. At run time, two processes, namely, *model assessment* and *model substitution*, continuously monitor the non-functional property(ies) of all models and apply model substitution when necessary to maintain stable application behavior. The two processes work in an *evaluation window*.

Let  $cl$  denote the set of candidate models  $\{m_0, \dots, m_k\}$  and  $\hat{m}$  the model currently in use. Process *model assessment* (Section 4) evaluates models in  $cl$  according to the given non-functional property  $p$ . It implements a Dynamic Multi-Armed Bandit (*Dynamic MAB*) approach, which extends our previous work built on traditional MAB [22] to continuously evaluate the models.

Process *model substitution* (Section 5) takes as input the results of process model assessment and selects the best model  $\hat{m}$  to be used within the application according to two strategies. The first strategy compares models in  $cl$  using the Dynamic MAB in the entire evaluation window, producing a model ranking. The best model in the ranking is then selected as the new  $\hat{m}$  to be used by the application in the following evaluation window. The second strategy extends the first one by implementing *early substitutions* of  $\hat{m}$  according to metric assurance level  $al$ , measuring the model degradation. Early substitutions anticipate the replacement of  $\hat{m}$ , addressing transient changes before the end of the evaluation window.

**Example 1 (Reference Scenario).** Our reference scenario considers an ML-based application that supports authorities (i.e., courts) in estimating the bail of an individual in

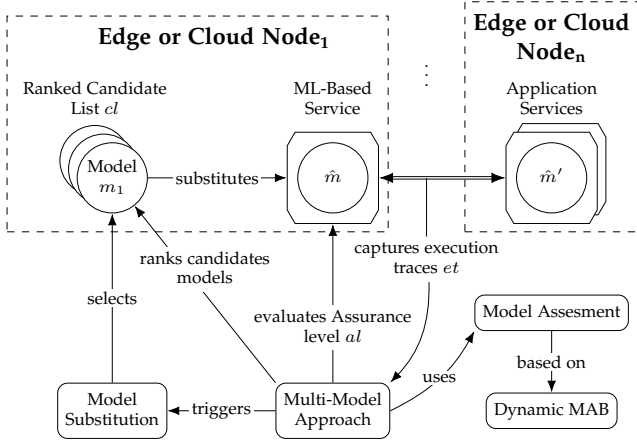


Fig. 1. Overview of our approach.

prison. The application trains 5 models  $cl = \{m_1, \dots, m_5\}$  in the cloud on the same dataset, containing data on past bails at national level. Each court is then provided with a model. Let us assume that the selected model is  $m_3$  (i.e.,  $\hat{m} = m_3$ ). Due to the nature of the task, the non-functional property of interest is *fairness*, in terms of variance over some protected attributes [22].

Let us assume that, at run time,  $\hat{m}$  shows significant biases in the presence of underrepresented/disadvantaged groups, thus affecting predicted bails. The overall fairness of  $\hat{m}$  must be evaluated and compared to the other candidate models, and a model substitution triggered when needed to maintain stable non-functional behavior.

Our reference scenario exemplifies the four main challenges of modern ML applications: *i*) the definition of advanced non-functional properties that are typical of ML such as fairness and privacy; *ii*) the assessment and comparison of models in terms of a given non-functional property; *iii*) the detection of an application's non-functional property degradation at run time; *iv*) the automatic substitution of models to keep the application behavior stable with respect to their non-functional properties. Existing solutions in literature cannot tackle these challenges in their entirety. For instance, QoS-aware service selection approaches (e.g., [4], [5], [30]) maximize specific (non-)functional metrics to build an optimum composition or retrieve the most suitable models. Similarly, classifier selection approaches (e.g., [12], [13], [14], [15]) maximize quality metrics such as accuracy, continuously swapping the models and potentially introducing fluctuations in the non-functional behavior. Other approaches target non-AI systems (e.g., [31]), or do not generalize over the ML algorithms or non-functional properties (e.g., [10], [11], [32], [33], [34]). To the best of our knowledge, our multi-model approach is the first solution that guarantees *stable application non-functional behavior over time* and is *generic with respect to the ML algorithm and property*. A detailed comparison of the approach in this paper with solutions in literature is provided in Section 8.

### 3 BUILDING BLOCKS

Our multi-model approach is based on three main building blocks: *i*) *execution traces* (Section 3.1), *ii*) *non-functional prop-*

TABLE 1  
Terminology.

| Symbol                           | Description   |
|----------------------------------|---|
| $m$                              | ML model  |
| $cl$                             | Set of candidate ML models  |
| $\hat{m}$                        | Selected ML model   |
| $et_t$                           | $t$ -th execution trace   |
| $p$                              | Non-functional property   |
| $p.\hat{p}$                      | Non-functional property name  |
| $p.S$                            | Non-functional property score function  |
| $m^*$                            | ML model providing the highest reward according to the MAB                        |
| $w$                              | Variable-sized sequence of observed execution traces (window)                     |
| $Beta_{m,j}$                     | Beta distribution of model $m$ in window $w_j$                                    |
| $\alpha_{m,j}$ and $\beta_{m,j}$ | Parameters of $Beta_{m,j}$  |
| $v_m(\theta)$                    | Reward retrieved from $Beta_{m,j}$  |
| $residual_r$                     | Threshold of residual value in the MAB for closing the window $w$                 |
| $\delta$                         | Memory to re-initialize $Beta_{m,j}$ in a new window $w$                          |
| $rm_{m,t}$                       | Value of ranking metric $rm$ of $m$ at the $t$ -th execution trace                |
| $al_t$                           | Assurance level of $\hat{m}$ at the $t$ -th execution trace                       |
| $deg_t$                          | Degradation of the assurance level of $\hat{m}$ up to the $t$ -th execution trace |
| $thr$                            | Threshold of assurance level degradation triggering early substitution            |

*erties* (Section 3.2), and *iii*) *Multi-Armed Bandit* (Section 3.3). Table 1 shows the terminology used in this paper.

#### 3.1 Execution Traces

Execution traces capture the behavior of a given ML model at run time. They can be defined as follows.

**Definition 1 (Execution Trace).** An execution trace  $et$  is a tuple of the form  $\langle dp, pred \rangle$  where *i*)  $dp$  is the data point (i.e., a set of features) given as input to a model, *ii*)  $pred$  is the predicted result.

We note that  $dp$  can also contain the raw samples given as input to a deep learning model. Execution traces can be captured, for instance, by intercepting calls to the ML-based application or through monitoring [35].

**Example 2 (Execution Trace).** Following Example 1, let us consider an execution trace  $et = \langle [age=27, gender=male, race=latino, past-offence=0, \dots], \$10K \rangle$ , retrieved by monitoring model  $\hat{m}$ , where:  $[age=27, \dots, ]$  is the data point sent to  $\hat{m}$  and  $\$10K$  is the predicted bail.

#### 3.2 Non-Functional Properties

Traditional non-functional properties are defined as an abstract property (i.e., the property name) refined by a set of attributes [23]. Common properties include *performance*, *confidentiality*, *integrity*, *availability* [36]. When an ML model is considered, the notion of property is redesigned [7] as follows.

**Definition 2 (Non-Functional Property).** A non-functional property  $p$  is a pair  $p = (\hat{p}, S)$ , where  $\hat{p}$  is an abstract property taken from a shared controlled vocabulary [22] and  $S$  is a score function of the form  $S: \{et\} \rightarrow \mathbb{R}$  quantitatively describing *how much* an ML model supports  $\hat{p}$  according to its execution traces.

In the following, we use the dotted notation to refer to the components of  $p$  (e.g.,  $p.S$ ).

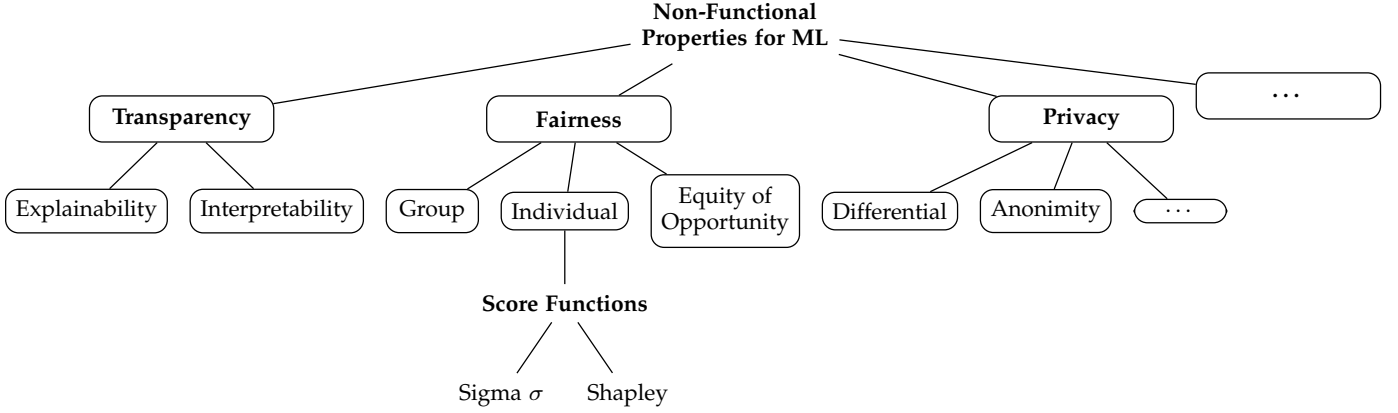


Fig. 2. Partial view of the ML property taxonomy [22].

**Example 3 (Non-Functional Property).** Following Example 2, property *fairness* can be defined as  $p_{\text{fairness}} = (\text{fairness}, \text{variance-over-gender-race})$ , where the score function *i*) generates a number of synthetic data points *dp* covering all the possible combinations of protected attributes *gender* and *race*; *ii*) sends each *dp* to the model; and *iii*) measures the variance  $\sigma^2$  over the predicted bails. We note that the higher the variance, the lower the support for property fairness.

Non-functional properties of ML can be peculiar properties purposefully defined for ML evaluation (e.g., adversarial robustness [21]) or a new interpretation of traditional ones (e.g., model prediction integrity) [7]. Figure 2 shows a portion of our taxonomy of non-functional properties, which has been fully presented in our previous work [22]. The taxonomy includes generic properties, which are then refined by detailed properties. For example, *transparency* is a generic property with two sub-properties: *i*) *explainability*, the capability to explain the model, on one hand, and individual decisions taken by the model, on the other hand; and *ii*) *interpretability*, the capability to predict the consequences on a model when changes are observed. As another example, *fairness* is a generic property with multiple sub-properties. For each detailed property, different score functions can be defined. For instance, Figure 2 shows two score functions for property *individual fairness*: variance  $\sigma$  (used in this paper) and *Shapley* [37]. Score functions in the taxonomy are general, though we note that they need to be refined and instantiated in the context of an evaluation process for a specific ML-based application.

### 3.3 The MAB

We use the Multi-Armed Bandit (MAB) technique [22] to compare models according to a non-functional property *p* on a set of execution traces. MAB repeatedly executes an experiment, whose goal is to get the highest reward that can be earned by executing a specific action chosen among a set of alternatives. Every action returns a reward or a penalty with different (and unknown) probabilities. The experiment is commonly associated with the problem of a gambler facing different slot machines (or a single slot machine with many arms retrieving different results). In our scenario, the

actions are the models  $m_i$  in the candidate list *cl* and the reward is based on the score function  $p.S$  in Definition 2.

**Definition 3 (MAB).** Let *cl* be the set of candidate models  $\{m_1, \dots, m_k\}$ , each associated with an unknown reward  $v_m$  for non-functional property *p*. The goal of the MAB is to select the model  $m^*$  providing the highest reward in a set of experiments (i.e., a set of execution traces). A probability distribution  $f_m(y | \theta)$  drives experiments' rewards, with *y* the observed reward and  $\theta$  a collection of unknown parameters that must be learned through experimentation. MAB is based on Bayesian inference considering that, in each experiment, the success/failure odd of each model is unknown and can be shaped with the probability distribution *Beta distribution*. Let *m* be a model, its Beta distribution  $\text{Beta}_m$  is based on two parameters  $\alpha, \beta \in [0, 1]$  (denoted as  $\alpha_m$  and  $\beta_m$ , resp.) and its probability density function can be represented as

$$\text{Beta}_m(x; \alpha_m, \beta_m) = \frac{x^{\alpha_m-1}(1-x)^{\beta_m-1}}{B(\alpha_m, \beta_m)} \quad (1)$$

where the normalization function *B* is the Euler beta function

$$B(\alpha_m, \beta_m) = \int_0^1 x^{\alpha_m-1}(1-x)^{\beta_m-1} dx \quad (2)$$

Thompson sampling [38] pulls models in *cl*, as a new trace *et* is received from the application, by sampling the models' Beta distribution. The model with the highest sampled reward (denoted as  $m^*$ ) is then evaluated according to  $p.S$  and *et*. A comparison of the score function output against a threshold determines the success or failure of this evaluation.  $\text{Beta}_{m^*}$  is then updated accordingly, such that  $m^*$  is *pulled* more frequently in case of successful evaluation ( $\alpha_{m^*}$  increased by 1), less frequently ( $\beta_{m^*}$  increased by 1), otherwise.

Let  $y_t$  denote the set of observations recorded up to the *t*-th execution trace  $et_t$ . The optimal model  $m^*$  is selected according to probability  $\text{winner}_{m,t}$ :

$$\begin{aligned} \text{winner}_{m,t} &= P(m^* | y_t) = \\ &= \int l(m^* = \arg \max_{m \in cl} v_m(\theta)) p(\theta | y_t) dt \end{aligned} \quad (3)$$

where  $l$  is the indicator function and  $p(\theta|y_t)$  is the Bayesian posterior probability distribution of  $\theta$  given the observations up to the  $t$ -th execution trace. The MAB terminates when all experiments end, that is, all traces have been received.

The optimal model  $m^*$  is used by the application (i.e.,  $\hat{m}=m^*$ ) [22]. We note that, while effective at application startup, the MAB cannot be continuously applied at run time as new traces come. For this reason, the MAB in this section (*Static MAB* in the following) is only used for static model selection at development time. We then define in Section 4 a *Dynamic MAB* as the extension of the Static MAB for run-time model selection and substitution.

#### 4 MODEL ASSESSMENT: DYNAMIC MAB

Process model assessment compares ML models at run time according to their non-functional behavior. It takes as input the models in the candidate list  $cl$  and the non-functional property  $p$ , and returns as output the models' Beta distributions. Models assessment uses the Static MAB within an *evaluation window*  $w$  of  $|w|$  execution traces, and then shifts the window of  $|w|$  execution traces instantiating a new Static MAB.

The window size  $|w|$  can be fixed or variable. When  $w$  has fixed size  $|w|$ , the Dynamic MAB may not reach a statistical relevance to take a decision; in this case, *i*) the outcome can be sub-optimal or *ii*) the evaluation can be extended to the next window.

When  $w$  has variable size  $|w|$ , our default approach, the MAB terminates the evaluation and moves to the next window only when a statistically relevant decision can be made. It is based on the *value remaining in the experiment* [39], a tunable strategy that controls both the estimation error and the window size requested to reach a valuable decision. In the following, we present our solutions based on variable window sizes, namely Dynamic MAB with Variable Window (DMVW) and DMVW with Memory (DMVW-Mem).

##### 4.1 Dynamic MAB with Variable Window (DMVW)

The Dynamic MAB with Variable Window (DMVW) implements the *value remaining in the experiment* using a Monte Carlo simulation. The simulation considers a random set  $g$  of sampled draws from models' Beta distributions. It then counts the frequency of each model being the winner in  $g$  as an estimation of the corresponding probability distribution.

The *value remaining in the experiment* is based on the minimization of the "regret" (the missed reward) due to an early terminated experiment. Let  $\theta_0$  denote the value of  $\theta$  and  $m^*=\arg\max_{m\in cl} v_m(\theta_0)$  the optimal model at the end of a window  $w$ . The regret due to early termination of an experiment within window  $w$  is represented by  $v_{m^*}(\theta_0)-v_{m^*,t}(\theta_0)$ , which is the difference between *i*) the reward  $v_{m^*}(\theta_0)$  of the optimal model  $m^*$  retrieved at the end of window  $w$  and *ii*) the reward  $v_{m^*,t}(\theta_0)$  of the optimal model  $m^*,t$  retrieved at execution trace  $et_t$ .

Considering that the regret is not directly observable, it can be computed using the posterior probability distribution. Let us consider  $v_*(\theta^{(g)})=\max_{m\in cl} v_m(\theta^{(g)})$  where  $\theta^{(g)}$  is drawn from  $p(\theta|y_t)$ . The "regret"  $r$  in  $g$

is  $r^{(g)}=v_*(\theta^{(g)})-v_{m^*,t}(\theta_0)$ , which derives from the regret posterior probability distribution. We note that  $v_*(\theta^{(g)})$  is the maximum available value within each Monte Carlo draw set  $g$  and  $v_{m^*,t}(\theta^{(g)})$  is the value (alike taken in  $g$ ) for the best arm within each Monte Carlo simulation. Regret is expressed as the percentage of the deviation from the model identified as the winner, so that draws from the posterior probability are given as follows.

$$r^{(g)} = \frac{v_*(\theta^{(g)}) - v_{m^*,t}(\theta^{(g)})}{v_{m^*,t}(\theta^{(g)})} \quad (4)$$

The experiment completes when 95% of the samples of a simulation have a residual value less than a given percentage (*residual<sub>r</sub>*) of the value of the best model  $v_{m^*,t}(\theta_0)$ . Formally, a window can be closed when  $\text{percentile}(r^{(g)}, 95) \leq v_{m^*,t}(\theta_0) \times \text{residual}_r$ . A common value for *residual<sub>r</sub>* is 1%; it can be increased to reduce the window size, while leading to a greater residual. We note that the window size can be tuned in terms of the acceptable regret using *residual<sub>r</sub>*.

In a nutshell, DMVW takes a decision based on the execution traces in a specific window  $w$  only. A new MAB is executed from scratch in each window, potentially leading to a discontinuous model comparison. Due to this effect, DMVW can produce fluctuations in the selection of the optimal model  $m^*$  to be used by the application. To address these issues, we extend the DMVW with the notion of *memory* in Section 4.2.

##### 4.2 DMVW with Memory (DMVW-Mem)

The *DMVW with memory* (DMVW-Mem) keeps track of past DMVW executions to smooth the discontinuity among consecutive windows. DMVW-Mem for window  $w_j$  is defined on the basis of the Beta distributions and corresponding parameters in window  $w_{j-1}$  as follows.

**Definition 4 (DMVW with memory (DMVW-Mem)).** A DMVW-Mem is a DMVW where the Beta distribution  $\text{Beta}_{m,j}$  of each model  $m$  in window  $w_j$  is initialized on the basis of the Beta distribution  $\text{Beta}_{m,j-1}$  of the corresponding model  $m$  in window  $w_{j-1}$ , as follows:

- $\alpha_{m,j}=\alpha_{m,j-1} \times \delta$
- $\beta_{m,j}=\beta_{m,j-1} \times \delta$ ,

where  $\delta \in [0, 1]$  denotes the memory size,  $\alpha_{m,j-1}$  and  $\beta_{m,j-1}$  are  $\alpha$  and  $\beta$  of Beta distribution  $\text{Beta}_{m,j-1}$  of model  $m$  in window  $w_{j-1}$ . We note that the resulting  $\alpha_{m,j}$  and  $\beta_{m,j}$  are rounded down and set to 1 when equal to 0.

DMVW-Mem initializes the Beta distributions in each window  $w_j$  according to the Beta distribution parameters observed in window  $w_{j-1}$ .

**Example 4.** Following Example 3, let us assume that the current evaluation window  $w_{11}$  in a given court terminates after 200 execution traces according to DMVW-Mem. The output of process model assessment is  $\{\text{Beta}_{m_1,11}, \dots, \text{Beta}_{m_5,11}\}$ . Figure 3 shows  $\text{Beta}_{m_5,11}$ , where  $\alpha_{m_5,11}=110$  and  $\beta_{m_5,11}=2$ , meaning that  $m_5$  has been frequently sampled and successfully evaluated. Let us then assume that the memory has size 10% (i.e.,  $\delta=0.1$ ). Figure 3 shows  $\text{Beta}_{m_5,12}$  defined for window

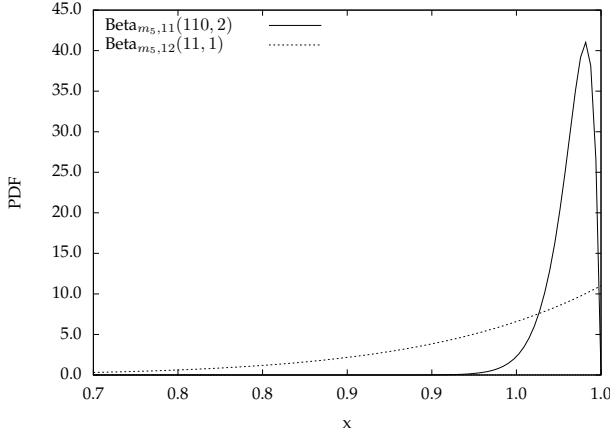


Fig. 3. Examples of Beta distributions.  $\text{Beta}_{m_5,11}(110, 2)$  is retrieved at the end of window  $w_{11}$ ,  $\text{Beta}_{m_5,12}(11, 1)$  is used for the next window  $w_{12}$ , with  $\delta=10\%$ .

$w_{12}$ , which is initialized as: i)  $\alpha_{m_5,12}=110 \times 0.1=11$ ; ii)  $\beta_{m_5,12}=2 \times 0.1=0.2$ , which is then set to 1 according to Definition 4.

## 5 MODEL SUBSTITUTION

Process model substitution is executed on the basis of process model assessment in Section 4. It takes as input the results of the DMVW-Mem evaluation in the current window, and returns as output the model to be selected and used by the application in the following window.

### 5.1 Ranking-Based Substitution

Ranking-based substitution ranks models  $m \in cl$  in a given window  $w_j$  and determines the model  $\hat{m}$  to be used in the following window  $w_{j+1}$ . Let us recall that  $\alpha_m$  ( $\beta_m$ , resp.) is incremented by 1 when  $p.S$  is successfully (unsuccessfully, resp.) evaluated on trace  $et \in w_j$  (Section 3.3). Ranking-based substitution is based on a metric evaluating how frequently each model is selected by Thompson Sampling and successfully evaluated in DMVW-Mem, as follows.

**Definition 5 (Ranking Metric).** Let  $w_j$  be a window and  $m$  a model. The value of ranking metric  $rm_{m,j}$  of  $m$  in  $w_j$  is retrieved as  $\alpha_{m,j}/(\alpha_{m,j} + \beta_{m,j})$ .

According to Definition 5,  $rm_{m,j}$  is the ratio between the number of successful evaluations of  $m$  (in terms of  $p.S$ ) and the total number of draws computed by DMVW-Mem in  $w_j$ . It is retrieved for every model in  $cl$  and used for ranking.

**Substitution.** At the end of window  $w_j$ , the top-ranked model  $\hat{m}$  is selected and used within window  $w_{j+1}$ . The substitution happens when  $w_j$  terminates according to the value remaining in the experiment (Section 4.1).

We note that the ranking can also be used in case further substitutions in  $w_{j+1}$  are needed. For instance, the second model in the ranking is used when  $\hat{m}$  experiences an (unrecoverable) error.

**Example 5.** Following Example 4, the ranking metric  $rm_{m_5,11}$  has value  $110/(110 + 2) \approx 0.98$ . Let us assume that  $rm_{m_5,11}$  has the highest value:  $m_5$  substitutes the

model used in  $w_{11}$ , and is used for bail prediction in  $w_{12}$  (i.e.,  $\hat{m}=m_5$ ).

The assumption that the ranking computed for  $w_j$  is appropriate for  $w_{j+1}$  does not hold when transient changes in the models non-functional behavior are observed within  $w_{j+1}$  (e.g., a sharp change in the environmental context). In this scenario, although  $\hat{m}$  becomes suboptimal, it cannot be substituted until the following window begins. To address this issue, we propose an approach based on early substitution that is presented in Section 5.2.

### 5.2 Assurance-Based Substitution

Assurance-based substitution triggers early substitution of the selected model  $\hat{m}$  before window  $w$  terminates. It monitors  $\hat{m}$  by computing its *assurance level* as follows.

**Definition 6 (Assurance level).** Let  $\hat{m}$  be the selected model and  $et_t \in w_j$  an execution trace. The *assurance level*  $al_t$  of  $\hat{m}$  given  $et_t$  is  $v_{\hat{m}^t}(\theta)/v_*(\theta^{(g)})$ .

According to Definition 6,  $al_t$  is the ratio between i) the reward  $v_{\hat{m}^t}(\theta)$  of the selected model  $\hat{m}$  retrieved at execution trace  $et_t$  and ii) the reward  $v_*(\theta^{(g)})$  of the optimal model  $m^*$ , according to the Monte Carlo simulation in DMVW-Mem (Section 4.1). We note that the assurance level can be retrieved for each model  $m_i$  using the corresponding reward as numerator.

The assurance level  $al$  is used to calculate the degradation of the selected model. Formally, let  $et_t$  be an execution trace in window  $w_j$ . The *degradation* of  $\hat{m}$  at  $et_t \in w_j$  is defined as follows.

$$\text{deg}_t = 1 - \frac{\sum_{i=1}^t al_i}{t} \quad (5)$$

**Substitution.** It works as the ranking-based substitution but the selected model  $\hat{m}$  is substituted with the second model in the ranking before the window termination (i.e., early substitution), *iff* its degradation  $\text{deg}_t$  exceeds threshold  $thr$  ( $\text{deg}_t > thr$ ).

Early substitution copes with transient changes within the window according to the degradation represented in  $thr$ . A high (low, resp.) threshold means high (low, resp.) tolerance. For instance, a high tolerance is preferable when the substitution overhead is high (e.g., when large models should be physically moved). A low tolerance is preferable when small variations in the properties of the deployed models has a strong impact on the application behavior. We note that, given its fundamental role in the substitution process, we experimentally evaluated the adoption of different degradation thresholds  $thr$  in Section 7.3.

**Example 6.** Following Example 5, let us consider model  $m_5$  as the selected model and model  $m_4$  as the second model in the ranking. Figure 4 shows an example of the assurance levels of  $\hat{m}_5$  and  $m_4$ , denoted as  $al_{\hat{m}_5,t}$  and  $al_{m_4,t}$ , respectively. Figure 4 also shows the corresponding logarithmic trend lines for readability, and the value remaining in the experiment, using DMVW-Mem.

Let us first consider ranking-based substitution only. At  $t=290$ , window  $w_j$  terminates according to the value remaining in the experiment (Section 4.1). DMVW-Mem

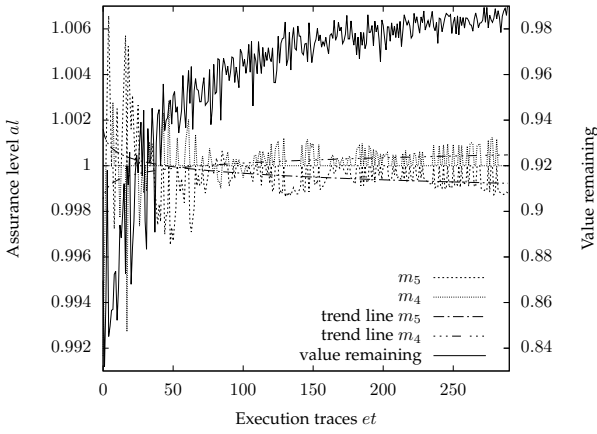


Fig. 4. Assurance levels of  $\hat{m}_5$  (ranked first) and  $m_4$  (ranked second), denoted as  $al_{\hat{m}_5,t}$  and  $al_{m_4,t}$ , respectively. The plot shows the logarithmic trend lines and the outcomes in relation to the value remaining in the experiment of the DMVW-Mem in a given time window  $w$ .

recomputes the ranking;  $m_4$  is the top-ranked model while  $m_5$  the second one. DMVW-Mem triggers ranking-based substitution, and  $m_4$  becomes the selected model ( $\hat{m}=m_4$ ) for window  $w_{j+1}$ .

Let us then consider assurance-based substitution. We can observe that  $al_{\hat{m}_5,t}$  decreases as execution traces arrive. From  $t=38$ ,  $al_{\hat{m}_5,t}$  stably becomes less than 1. Around  $t=87$ ,  $al_{m_4,t}$  overcomes  $al_{\hat{m}_5,t}$  thus suggesting a possible substitution. However, the degradation of model  $m_5$  is not severe enough to justify the early substitution (i.e., the degradation is lower than the degradation threshold).

## 6 WALKTHROUGH

We present a walkthrough of our approach based on the reference scenario in Example 1. Figure 5 shows the pseudocode of our approach.

The five models in the candidate list  $cl$  in Example 1 are first evaluated offline using the Static MAB in Section 3.3, to retrieve the optimum model  $m^*$  that initializes our approach. Let us assume that model  $m_2$  is selected as the optimal model ( $\hat{m}=m^*$ ). Our model assessment and substitution processes (Figure 5) begin, instantiating the DMVW-Mem. The processes take as input *i*) the models in  $cl$ , *ii*) the observed execution traces, *iii*) the non-functional property fairness in Example 3, *iv*) the memory size  $\delta$ , *v*) the early substitution threshold  $thr$ , and *vi*) the minimum number of MAB iterations. Execution traces observed from all the models are given as input to process model assessment. For each execution trace, the function **thompson\_sampling** in DMVW-Mem chooses a model among those in  $cl$  by drawing a sample from each model Beta distribution and retrieving the one with the highest value. The retrieved model is evaluated according to the fairness score function (function **score\_function** in Figure 6), updating the corresponding Beta distribution accordingly (Definition 4). Then, process model assessment invokes function **monte\_carlo\_simulation** to simulate the probabilities of models being winners. It creates a two-dimensional matrix with

dimensions  $|cl| \times g$ , where  $g$  is the number of estimations. Each cell contains samples drawn from the models' Beta distributions; the matrix counts the frequency of each model being winner and approximates the probability distribution  $p(\theta|y_t)$ , accordingly.

Process model assessment proceeds until the minimum number of iterations is met and the value remaining in the experiment permits to reach a statistically relevant decision (function **should\_terminate**). At this point, the evaluation window ends (function **handle\_window**).

Upon process model assessment ends, process model substitution invokes function **send\_into\_production** ranking models proportionally to the number of their successful evaluations of the non-functional property (ranking metric in Definition 5). For instance, the ranking at the end of window  $w_1$  is  $\{m_3, m_2, m_1, m_4, m_5\}$ , from best to worst. The top-ranked model ( $m_3$ ) is pushed to production replacing  $m_2$  selected at deployment time by the Static MAB (Section 5.1). Process model substitution also monitors the selected model invoking function **assurance\_management**. The latter verifies whether the non-functional behavior of the selected model is worsening with respect to the optimum model estimated by the Monte Carlo simulation. It computes the assurance level  $al_t$  for each new execution trace  $et_t$  (Definition 6) and uses it to retrieve the overall degradation (equation 5). For instance, during window  $w_2$ , the degradation of  $m_3$  is negligible, meaning that  $m_3$  is still adequate according to the data observed in  $w_2$  and does not need to be substituted in advance.

When the current window  $w_2$  terminates, process model substitution recomputes the ranking. For instance, the ranking is  $\{m_3, m_2, m_4, m_1, m_5\}$ , and  $m_3$  is used as the selected model for window  $w_3$ . In  $w_3$ , process model substitution observes a constant degradation in the assurance level of  $m_3$ , reaching the early substitution threshold. Early substitution is therefore triggered and the second model in the ranking ( $m_2$ ) substitutes  $m_3$ . Again when the current window  $w_3$  terminates, process model substitution recomputes the ranking and  $m_2$  is confirmed at the top of the ranking.

Overall, this adaptive approach ensures that *i*) model substitution happens only when the decision is statistically relevant according to the observed behavior (*ranking-based substitution* in DMVW-Mem), *ii*) a sub-optimal substitution decision can be fixed as soon as it is detected without waiting for the entire evaluation window (*assurance-based substitution*), and *iii*) the entire process can be fine-tuned according to each scenario.

## 7 EXPERIMENTAL EVALUATION

We experimentally evaluated our approach focusing on: *i*) the model assessment at development time using Static MAB; *ii*) the model substitution at run time using Dynamic MAB, also evaluating the impact of different memory sizes; *iii*) quality and *iv*) performance of ranking-based and assurance-based substitutions.

### 7.1 Experimental Settings

We considered the application for bail estimation and property fairness in our reference example in Section 6. In our

```

INPUT
cl: models to be ranked
et[: execution traces
p: target non-functional property
δ: memory size
thr: early substitution threshold
burn_in: minimum number of iterations

MAIN
i=0 /* Iteration counter */

/* Begin the DMVW-Mem */
for each ett ∈ et[]
  m* = thompson_sampling(cl)
  apply_score_function(ett, m*,
                        VARIANCETHRESHOLD)
  (g[], p(θ|yt)) = monte_carlo_simulation(cl)
  est_models[:] = record current estimates of
  each m ∈ cl wins according to ranking metric
  if (i > burn_in) AND should_terminate(p(θ|yt),
                                           est_models, g[])
    handle_window()
  else
    i = i + 1
  /* Continue with the next trace */

THOMPSON_SAMPLING
for each m ∈ cl
  sample = draw sample from Betam
  m* = arg max sample
  return m*

MONTE_CARLO_SIMULATION
g = 100
g[:] = matrix |cl| × g of samples
  drawn from Betam ∀ m ∈ cl
  compute p(θ | yt) from g[]
  return g[], p(θ|yt)

SHOULD_TERMINATE
residualr = 0.01
  compute vm*,t(θ0) from est_models[]
   $r(g) = \frac{v_*(\theta^{(g)}) - v_{m^*,t}(\theta^{(g)})}{v_{m^*,t}(\theta^{(g)})}$ 
  return percentile(r(g), 0.95) ≤ vm*,t(θ0) ×
  residualr

HANDLE_WINDOW
send_into_production(m*, cl)
j - 1, j = previous and current windows
for each m ∈ cl
  /* Initialize Beta distrib. according to δ */
  αm,j = αm,j-1 × δ
  βm,j = βm,j-1 × δ

SEND_INTO_PRODUCTION
models_rank = sort cl in descending order
  according to αm / (αm + βm)
assurance_management(models_rank)
m̂ = models_rank[0]
push(m̂) into production

ASSURANCE_MANAGEMENT
cum_assurance = 0
i = 1 /* counter */
for each ett ∈ w
  alt = vm̂,t(θ) / v*(θ(g))
  cum_assurance = cum_assurance + alt
  degt = 1 - cum_assurance / i
  if degt > thr
    /* Early substitution */
    m̂ = models_rank[1]
    push(m̂) into production
  i = i + 1

```

Fig. 5. Pseudocode of our approach.

experiments, we used the dataset of the Connecticut State Department of Correction.<sup>1</sup> This dataset provides a daily updated list of people detained in the Department's facilities awaiting a trial. It anonymously discloses data of individual people detained in the correctional facilities every day starting from July 1<sup>st</sup>, 2016. It contains attributes such as last admission date, race, gender, age, type of offence and facility description, in more than four millions data points (at the download date). We divided this set into training and test sets, where the training set includes more than 3 million points.

We modeled the score function  $p.S$  of property fairness as the variance ( $\sigma^2$ ) of the bail amount in relation to sensitive attributes *gender* and *race* [40], [41], [42]. Figure 6 shows the pseudocode of the score function and its usage according to the threshold-based evaluation in Section 5.2. We generated five Naive Bayes models  $cl = \{m_1, \dots, m_5\}$ , each one trained on a training set randomly extracted from the main training set. The models showed similar performance, in terms of precision and recall in bail estimation. We also extracted 10 test sets corresponding to 10 individual experiments  $exp_1 - exp_{10}$  to be used in our experimental evaluation.

Experiments have been run on a laptop running Microsoft Windows 10, equipped with a CPU Intel Core i7 @ 2.6 GHz and 16 GBs of RAM, using Python 3 with libraries *numpy* v1.19.1 [43], *pandas* v1.2.5 [44], [45] and *scikit-learn* v0.22.1 [46]. Datasets, code, and experimental results are available at <https://github.com/SESARLab/continuous-management-of-ml-applications>.

## 7.2 Model Assessment

We present the experimental evaluation of our Static MAB for model assessment at development time. We compare the

1. Available at <https://data.ct.gov/Public-Safety/Accused-Pre-Trial-Inmates-in-Correctional-Facility/b674-676w> and downloaded on February 21<sup>st</sup>, 2020.

```

INPUT
et: execution trace
m*: Thompson selected model
VARIANCETHRESHOLD: variance threshold

```

```

OUTPUT
αm* and βm*

```

```

SCORE_FUNCTION
rows[:] = et
rows[:] += generate test data for all protected
  groups against et
predicted_vals[:] = m*.predict(rows[])
var = variance(predicted_vals[])
return var

```

```

APPLY_SCORE_FUNCTION
var = score_function(et)
if (var < VARIANCETHRESHOLD)
  αm* = αm* + 1
else
  βm* = βm* + 1

```

Fig. 6. Pseudocode of the score function of property fairness and its usage.

five Naive Bayes models using the Static MAB approach, by evaluating their behavior with respect to non-functional property fairness. Table 2 shows the Thompson Sampling draws for the five models in the candidate list on a randomly chosen sample (2,000 data points) for each of the 10 experiments.

Table 2 shows the distribution of models selected as best candidate (denoted in bold) for property fairness. Since  $m_3$  is never selected as the best candidate, it is removed from the candidate list for the rest of the experimental evaluation. We note that comparing models based on the same algorithm (i.e., Naive Bayes) is more challenging than considering different algorithms [22], posing our experi-



TABLE 2

Static MAB comparison in terms of Thompson Sampling draws on a random sample of 2,000 data points for each experiment. The best candidate for each experiment is denoted in bold.

| Experiments | $m_1$        | $m_2$       | $m_3$ | $m_4$        | $m_5$        |
|-------------|--------------|-------------|-------|--------------|--------------|
| $exp_1$     | <b>1,021</b> | 156         | 609   | 28           | 186          |
| $exp_2$     | 4            | <b>876</b>  | 349   | 462          | 309          |
| $exp_3$     | 414          | 286         | 341   | <b>645</b>   | 314          |
| $exp_4$     | 55           | 198         | 670   | <b>1,028</b> | 49           |
| $exp_5$     | 432          | 84          | 208   | <b>666</b>   | 610          |
| $exp_6$     | 138          | 255         | 50    | 607          | <b>950</b>   |
| $exp_7$     | 419          | 31          | 268   | 514          | <b>768</b>   |
| $exp_8$     | 205          | 528         | 135   | 104          | <b>1,028</b> |
| $exp_9$     | 288          | 394         | 35    | 43           | <b>1,240</b> |
| $exp_{10}$  | 104          | <b>1278</b> | 112   | 453          | 53           |

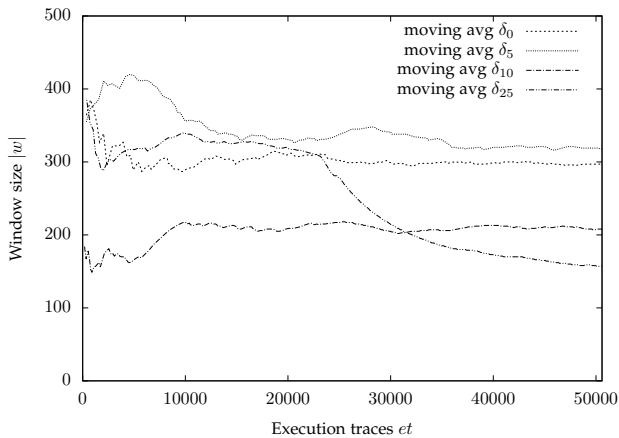


Fig. 7. Individual window sizes and moving average trends across all sets of execution traces with different memory sizes for  $exp_1$ .

ments in a worst-case scenario.

### 7.3 Model Substitution

We present the experimental evaluation of our process model substitution using DMVW-Mem with different memory sizes ( $\delta_0=0\%$ ,  $\delta_5=5\%$ ,  $\delta_{10}=10\%$ ,  $\delta_{25}=25\%$ ). We evaluated *i*) the impact of the memory on the window size, *ii*) the impact of the ranking-based substitution in terms of stability of model selections, *iii*) the quality of the ranking-based substitution, and *iv*) the quality of the assurance-based substitution. We note that no artificial degradation was introduced during the experiments.

#### 7.3.1 Memory Size and Ranking

Figure 7 shows the window size varying the memory in experiment  $exp_1$  with residual threshold  $residual_r=0.01$  (Section 4.1). We note that a bigger memory corresponds to a smaller window. This is expected, since the DMVW-Mem does not start from scratch in every window, and the more DMVW-Mem knows about the models' Beta distributions, the sooner the value remaining in the experiment reaches the threshold. Considering all the experiments, the average window size for  $\delta_{25}$  is 157 confirming the trend in Figure 7.

Let us now consider the model selected according to the DMVW-Mem ranking. Figure 8 shows the selected model

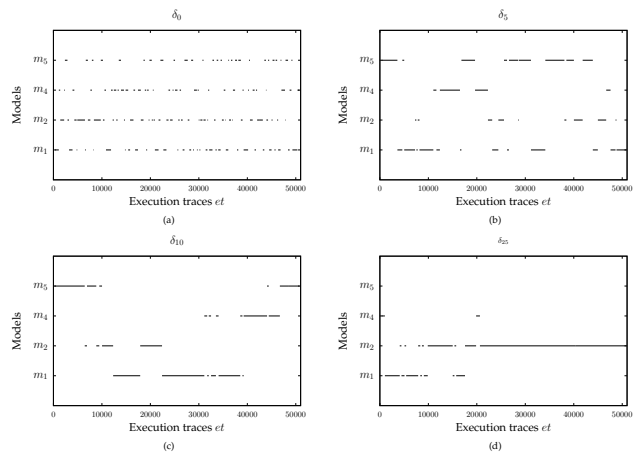


Fig. 8. The selected model for each execution trace  $et$  of experiment  $exp_1$  with different memory sizes  $\delta$ .

for each set of execution traces in experiment  $exp_1$ , considering different memory sizes. We note that extemporaneous changes on the selected model are frequent without memory ( $\delta_0$ ), less frequent with  $\delta_5$ , where clusters of continuously selected models emerge, and highly infrequent with  $\delta_{10}$ . Figure 8(d) shows a stable selection of model  $m_2$ , while models  $m_4$  and  $m_5$  are often not selected preferring  $m_1$  instead. Considering the entire ranking, model  $m_1$  is ranked at the second position with  $\delta_{25}$ , while  $m_4$  at the third position.

In general, we observe that the number of changes across the experiments, in terms of selected models, decreases as the memory increases. On average, across all experiments, it decreases by 41.18% when memory increases from  $\delta_5$  to  $\delta_{10}$  (from 34 changes on average with  $\delta_5$  to 20 changes on average with  $\delta_{10}$ ); it decreases by 20% when memory increases from  $\delta_{10}$  to  $\delta_{25}$  (from 20 changes on average with  $\delta_{10}$  to 16 changes on average with  $\delta_{25}$ ).

Figure 9 shows an aggregated ranking for all the experiments with  $\delta_{10}$ . It shows the percentage of times a model has been ranked into a specific position for all the experiments. We note that  $exp_1$  was one of the most balanced experiments in terms of ranking, having at least three models ( $m_1$ ,  $m_2$ , and  $m_5$ ) with a similar percentage of first and second positions in the ranking. In  $exp_2$  and  $exp_8$ ,  $m_4$  and  $m_5$  were ranked at first or second position  $\approx 80\%$  of the times. More specifically,  $m_1$  and  $m_5$  are ranked as the first model in the ranking 41.39% and 27.87% of times, respectively (Figure 8(c)). When  $m_5$  is not ranked first, it is ranked second 26.36% of times, while  $m_1$  17.21%.

Considering all experiments and memory sizes, when compared with  $\delta_5$ , we note an average decrease of ranking changes in the first position of 34.66% with  $\delta_{10}$  and of 62.43% with  $\delta_{25}$ . This is also clear from Figure 8(d), where  $m_2$  was ranked in the first position most of the times.

#### 7.3.2 Quality Evaluation

We evaluated the quality of ranking-based substitution and assurance-based substitution varying the memory. The ranking retrieved according to DMVW is used as baseline.

Let  $R$  denote the function that returns as output the (current) position in the DMVW-based ranking of the (cur-

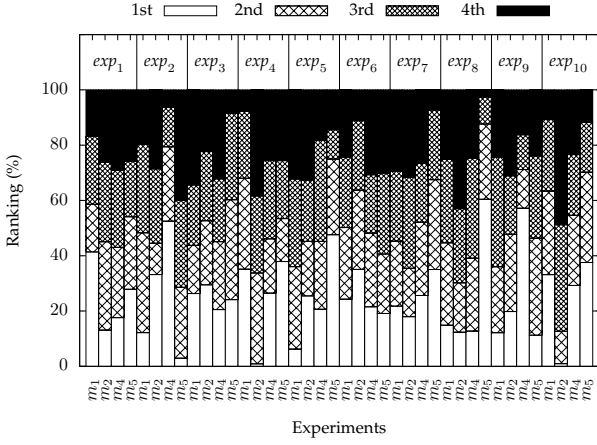


Fig. 9. Stacked histograms showing the ranking of the models in each of the 10 experiments with  $\delta_{10}$ .

rent) top-ranked model in the DMVW-Mem-based ranking; this position is a number  $\in [1, \dots, k]$  with  $k=|cl|$ . The *residual error*  $\xi$  measures the difference between the ranking obtained with DMVW-Mem and DMVW, and is defined as:

$$\xi = \text{penalty} \left( \frac{R-1}{k-1} \right), \quad (6)$$

where *penalty* is the *residual penalty function*. We note that in case the top-ranked model according to DMVW-Mem is top-ranked also according to DMVW, the residual error is  $\xi = \text{penalty}(0)$ ; in case it is ranked last according to DMVW, the residual error is  $\xi = \text{penalty}(1)$ . Residual penalty function *penalty* is defined as a sigmoid function as follows:

$$\text{penalty}(x) = \frac{1}{1 + e^{-c_1(x-c_2)}}, \quad (7)$$

where  $c_2$  control the  $x$  of the sigmoid inflection point and  $c_1$  the slope. The residual error measures the difference in terms of ranking between the different settings. While it is not an indicator of the absolute quality, we assume this measure as a valid indicator of the relative quality between the different settings of our solution.

**Ranking-based substitution:** Figure 10 shows the *cumulative residual error*  $\xi = \sum_t \xi$  (i.e., the sum of the error retrieved in each window and execution trace) for  $exp_1$  with different memory sizes. It also shows, marked with “×”, the execution traces where model substitutions occurred due to changes at the top of the ranking. We note that in this experiment the bigger the memory, the bigger the cumulative residual error. This effect is compensated by fewer model substitutions as also demonstrated in Section 7.3.1. We also note that, depending on the application domain, the memory settings can be dynamic. For instance, in scenarios where fast reaction to changes is more important than stability of the selected model, the memory can be lowered; it can be increased in scenarios where stability is important to counteract fluctuations.

Considering all experiments and memory sizes, we note an average cumulative residual error of 163.67 with  $\delta_5$  and of 243.63 with  $\delta_{25}$ , corresponding to an average increase of 48.85%. As depicted in Figure 9, our experiments revealed

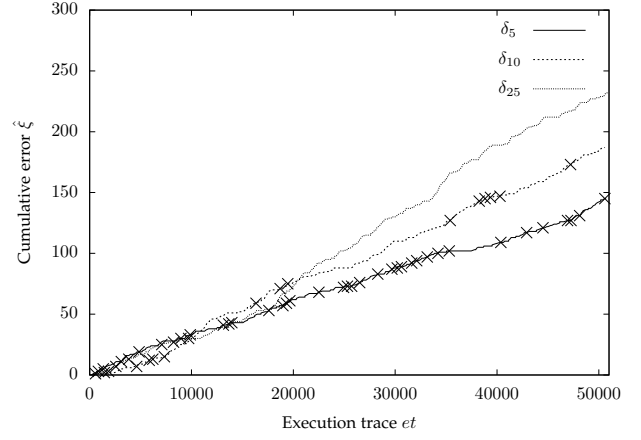


Fig. 10. Cumulative residual error  $\xi_k$  between DMVW ( $\delta_0$ ) and the DMVW-Mem with different memory sizes  $\delta_i$  and number of traces  $k$  for experiment  $exp_1$ . Model substitutions are marked with “X”.

a frequent variability among the best candidate models. Therefore, the most suitable approach in terms of residual error was the one having lower memory.

**Assurance-based substitution:** Using the memory settings in Section 7.3.1, we first evaluated the impact of the degradation threshold *thr*, varying its value in  $thr_5=0.05$ ,  $thr_{10}=0.10$ , and  $thr_{25}=0.25$ .

Figure 11 shows the total number of triggered early substitutions (denoted as *total*) compared with total number of substitutions that really occurred at end of the window (i.e, the correct early substitutions, denoted as *relevant*), on average across all experiments and memory settings. We observe that in 89% of the cases, an early substitution has been correctly triggered. In detail, an early substitution was correctly triggered in 81% of the cases when using  $thr_5$ , increasing to 92% when using  $thr_{25}$ . These results were expected, since a higher threshold corresponds to a more severe assurance variation, and thus to a higher likelihood of the change being correct at the end of the window.

Figure 11 also shows the number of successfully executed early substitutions (denoted as *success*) among the *relevant* early substitutions. A *successful* early substitution is a substitution where the model selected for substitution is the one evaluated by DMVW-Mem at the first position of the ranking at the end of the window. We observe that, in 93% of the cases on average, assurance-based early substitution took the correct decision. This result also confirms the quality of the entire retrieved ranking, meaning that when a substitution was needed, the second-ranked model was indeed the most suitable for substitution.

We also observe that *i)* as the early substitution threshold *thr* increases, the number of early substitutions decreases. For instance, with  $\delta_{10}$ , it decreases from 196 with  $thr_5$  to 147 with  $thr_{25}$ ; *ii)* the difference between the number of substitutions with  $thr_5$  and  $thr_{25}$  is lower than expected (e.g., from 301 to 243 with  $\delta_5$ ). In other words, when a degradation occurs, it exceeds  $thr_{25}$  in most of the cases. Even this experiment confirms that a bigger memory corresponds to fewer early substitutions. A more stable trend of the assurance level of the selected model was also observed

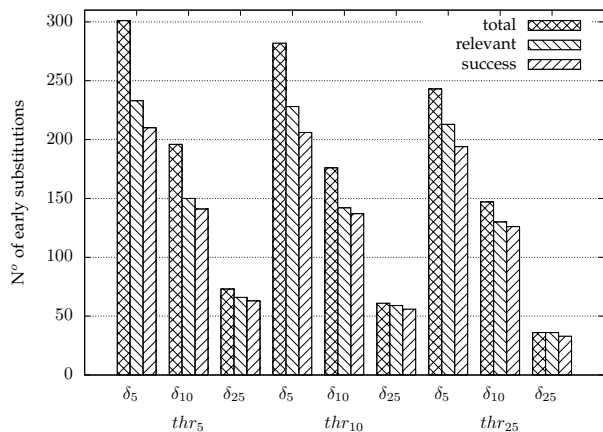


Fig. 11. Total number of triggered early substitutions (total), relevant early substitutions (relevant), and successful early substitutions (success) varying threshold  $thr$  and memory  $\delta$  for  $exp_1-exp_{10}$ .

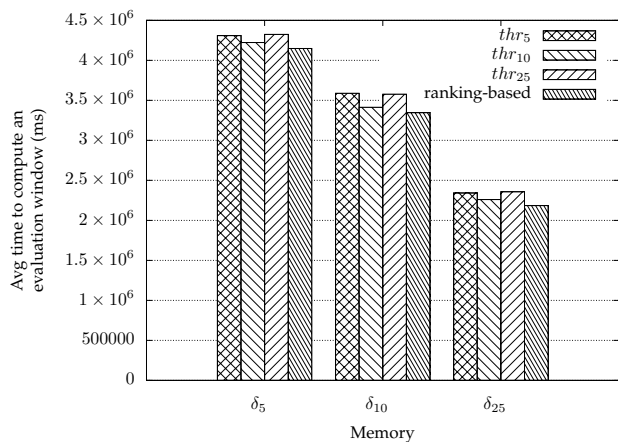


Fig. 12. Performance (average windows duration) expressed in milliseconds varying threshold  $thr$  and memory  $\delta$  for  $exp_1-exp_{12}$ .

with bigger memory.

We then evaluated the duration of early substitutions, in terms of the number of execution traces from the moment when the early substitution is triggered to the end of the considered window.

We observe that an increase in the memory and threshold results in a decrease in the number of substitutions (see Figure 11) and their duration. On average across all experiments, the duration varies from  $\approx 290$  execution traces when using  $\delta_5$  and  $thr_5$  to 84 when using  $\delta_{25}$  and  $thr_{25}$ .

Finally, we observe that, similarly to the memory tuning, the substitution threshold should be fine-tuned according to the different application domains, to adequately react to changes occurred within a given window.

**Performance:** We compared the performance of our ranking-based and assurance-based substitutions with different memory settings and assurance thresholds on all the experiments.

Figure 12 shows both the ranking-based and the assurance-based substitution performance varying memory settings and thresholds. The results are presented as the average time to compute an evaluation window. We

note that the ranking-based approach outperformed the assurance-based approach with an average improvement around 4.68%, due to the absence of assurance metric computations and corresponding comparisons. We also note that the impact of the different thresholds on performance is negligible, with  $thr_{10}$  showing the best performance in all conditions. Figure 12 clarifies that the dominating factor is the memory size. This is due to the fact that a bigger memory corresponds to a smaller window. In addition, it also corresponds to fewer substitutions positively impacting performance, because each substitution requires more iterations in DMVW-Mem to converge.

## 8 RELATED WORK

Our approach guarantees stable application behavior over time, by dynamically selecting the most suitable ML model according to a (set of) non-functional property. This issue has been studied from different angles in the literature: *i) classifier and ensemble selection*, *ii) functional and iii) non-functional ML adaptation*. At the end of this section, we also present a detailed comparison of our approach with the related work in terms of their category, objective, type of objective (functional/non functional), and applicability to ML models and properties.

**Classifier and ensemble selection** refers to the techniques that select the most suitable (set of) classifier among a set of candidates. It is referred to as *classifier selection* when one classifier is selected, *ensemble selection*, otherwise [51]. It can be performed at training time (*static*), or for each (subset of) data point at inference time (*dynamic*). The latter, often combined with static selection, typically shows the best performance [12]. Selection maximizes functional metrics, often accuracy. Meta-learning is frequently used, as surveyed by Khan et al. [52]. For instance, Cruz et al. [47] proposed a dynamic ensemble selection that considers different spatial-based criteria using a meta-classifier. Zhu et al. [14] defined a dynamic ensemble selection based on the generation of diversified classifiers. Selection is based on spatial information (i.e., the most *competent* classifiers for a region). Classifiers predictions are combined using weighted majority voting, weights depend on the classifiers competency for a data point. Zhang et al. [13] defined a dynamic ensemble selection whose selection criterion considers the *classifiers synergy*. It evaluates the *contribution* of each classifier to the ensemble, in terms of the accuracy retrieved with and without the classifier. For each data point, it selects the classifiers with a positive contribution, and uses such contribution as weight in predictions aggregation. Other approaches focused on imbalanced learning. Roy et al. [12] showed that specific preprocessing (e.g., oversampling of the underrepresented class) and dynamic, spatial-based selection outperform static selection in this scenario. Mousavi et al. [48] also used oversampling. Static selection then defines the ensemble and its combiner (e.g., majority voting). Dynamic selection finally retrieves a subset of the ensemble for each data point. Pérez-Gállego et al. [49] focused on *quantification tasks* with drifts between classes. The proposed dynamic ensemble selection uses a specifically designed criterion, selecting the classifiers whose training distribution is the most similar to the input data points. Our

TABLE 3  
Comparison with related work.

| Ref.                      | Cat. | Objective  | Objective Type | Applicability |          |
|---------------------------|------|--|----------------|---------------|----------|
|                           |      |  |                | ML Model      | Property |
| Cruz et al. [47]          | S    | Select the best ensemble for a (set of) data point wrt quality metrics                           | F              | ✓             | ✗        |
| Mousavi et al. [48]       | S    | Select the best ensemble for a (set of) data point wrt quality metrics                           | F              | ✓             | ✗        |
| Roy et al. [12]           | S    | Select the best ensemble for a (set of) data point wrt quality metrics                           | F              | ✓             | ✗        |
| Pérez-Gállego et al. [49] | S    | Select the best ensemble for a (set of) data point wrt quality metrics                           | F              | ✓             | ✗        |
| Zhang et al. [13]         | S    | Select the best ensemble for a (set of) data point wrt quality metrics                           | F              | ✓             | ✗        |
| Zhu et al. [14]           | S    | Select the best ensemble for a (set of) data point wrt quality metrics                           | F              | ✓             | ✗        |
| Almeida et al. [15]       | FA   | Select the best ensemble for a (set of) data point wrt quality metrics in the presence of drifts | F              | ✓             | ✗        |
| Tahmasbi et al. [9]       | FA   | Select the best classifier in the presence of drifts   | F              | ✓             | ✗        |
| Iosifidis et al. [32]     | NFA  | Fair predictions over time   | NF             | ✓             | ≈        |
| Zhang et al. [10], [11]   | NFA  | Balance quality and fairness over time   | F, NF          | ✗             | ≈        |
| Iosifidis et al. [34]     | NFA  | Fair predictions over time in online learning  | NF             | ≈             | ≈        |
| Zhang et al. [33]         | NFA  | Balance quality and fairness over time   | F, NF          | ✗             | ≈        |
| Badar et al. [50]         | NFA  | Train a fair, Federated Learning model   | NF             | ✗             | ≈        |
| This                      | -    | Stable application behavior over time wrt an arbitrary non-func. property                        | NF             | ✓             | ✓        |

approach implements a dynamic classifier selection, which departs from existing solutions implementing a (dynamic) selection of a (set of) classifier for each data point to maximize accuracy at inference time. Our goal is rather the *run-time selection and substitution of the ML model to the aim of guaranteeing a stable behavior of the application with respect to a specific (set of) non-functional property*.

**Functional adaptation** refers to the techniques that adapt a ML model (and application) according to changing conditions, notably a drift, to keep quality metrics high. According to the survey by Lu et al. [53], the possible actions upon a detected drift are: training and using a new ML model, using ensemble purposefully trained for drift, and adapting an existing ML model when the drift is localized to a region. The issue of drift has been approached using dynamic classifier selection. For instance, Almeida et al. [15] designed a drift detector whose selection criterion considers both spatial and concept-based information. It relies on a set of diverse classifiers that is dynamically updated, removing unnecessary classifiers and training new ones as new concepts emerge. Tahmasbi et al. [9] designed a novel adaptive ML model. It uses one classifier at time, and, upon drift detection, selects the subsequent classifier with the highest quality in the last evaluation window. Our approach implements an adaptation process, which departs from existing solutions based on the online re-training of individual ML models according to drift or the selection of ML models that maximize quality under drift. Our goal is rather the *adaptation of the overall ML-based application according to a (arbitrary) non-functional property of interest*.

**Non-functional adaptation** refers to the techniques that adapt a ML model (and application) according to a non-functional property. Fairness is the most studied property in literature in both static and dynamic settings; we focus on the latter due to its connection with the work in this paper. For instance, Iosifidis et al. [32] designed an approach that tackles fairness and concept drift. It uses two pre-processing techniques modifying data, which are then taken as input by classifiers that can natively adapt to concept drifts (e.g., *Hoefdding trees*). A similar solution is proposed

by Badar et al. [50] in federated learning. It first detects drift, and then evaluates if fairness is no longer supported. It then performs oversampling as countermeasure. Zhang et al. [10], [11] introduced a training algorithm based on *Hoefdding trees*, whose splitting criterion considers fairness and accuracy. Such idea has also been applied to random forest models [33]. Iosifidis et al. [34] designed an online learning algorithm that detects class imbalance and lack of fairness, and adjusts the ML model accordingly. It fixes weights during boosting (for imbalance) and the learned decision boundary (for fairness). Our approach implements an adaptation process, which departs from existing re-training solutions using a custom algorithm focused on a specific property (fairness). Our goal is rather the *adaptation of the overall application behavior according to any non-functional properties and ML algorithms*.

Table 3 shows how our approach compares with the related work in terms of *Category* (denoted as Cat.), *Objective*, *Objective Type*, and *Applicability*. *Category* can be *i*) classifier and ensemble selection (denoted as S), *ii*) functional adaptation (denoted as FA), and *iii*) non-functional adaptation (denoted as NFA). *Objective Type* can be *i*) functional (denoted as F), and *ii*) non-functional (denoted as NF). *Applicability* is expressed in terms of *i*) ML Model (✓ if applicable to any ML algorithm, ≈ if applicable to a class of ML algorithms, ✗ if applicable to a specific ML algorithm only); *ii*) Property (✓ if applicable to any (non-)functional property, ≈ if applicable to a class of (non-)functional properties, ✗ if applicable to a specific (non-)functional property). Table 3 shows that our approach (last row in Table 3) is the only *architectural and methodological solution that supports stable non-functional behavior of ML-based applications*. It builds on a smart and dynamic multi-model substitution departing from expensive re-training approaches and inference-time classifier selection for individual data points.

## 9 CONCLUSIONS

We presented a multi-model approach for the continuous management of ML-based application non-functional behavior. Our approach guarantees a stable application behav-

ior at run time, over time and across model changes, where multiple ML models with similar non-functional properties are available and one model is selected at time according to such properties and the application context. Our approach manages (dynamic and unpredictable) contextual changes in modern ML deployments, supporting early model substitutions based on Dynamic MAB and assurance evaluation.

## ACKNOWLEDGMENTS

Research supported, in parts, by *i)* project BA-PHERD, funded by the European Union – NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.1: “Fondo Bando PRIN 2022” (CUP G53D23002910006), *ii)* project MUSA – Multilayered Urban Sustainability Action – project, funded by the European Union - NextGenerationEU, under the National Recovery and Resilience Plan (NRRP) Mission 4 Component 2 Investment Line 1.5: Strengthening of research structures and creation of R&D “innovation ecosystems”, set up of “territorial leaders in R&D” (CUP G43C22001370007, Code ECS00000037); *iii)* project SERICS (PE00000014) under the NRRP MUR program funded by the EU – NextGenerationEU; *iv)* 1H-HUB and SOV-EDGE-HUB funded by Università degli Studi di Milano – PSR 2021/2022 – GSA – Linea 6; and *v)* program “Piano di Sostegno alla Ricerca” funded by Università degli Studi di Milano. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the Italian MUR. Neither the European Union nor the Italian MUR can be held responsible for them.

## REFERENCES

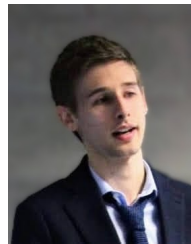
- [1] E. Damiani and C. Ardagna, “Certified Machine-Learning Models,” in *Proc. of SOFSEM 2020*, Limassol, Cyprus, January 2020.
- [2] T. L. Duc, R. G. Leiva, P. Casari, and P.-O. Östberg, “Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey,” *ACM CSUR*, vol. 52, no. 5, 2019.
- [3] European Union, “Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828 (Artificial Intelligence Act)Text with EEA relevance.” <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>, 2024.
- [4] K. Meng, Z. Wu, M. Bilal, X. Xia, and X. Xu, “Blockchain-enabled decentralized service selection for QoS-aware cloud manufacturing,” *Expert Systems*, 2024.
- [5] B. Qolomany, I. Mohammed, A. Al-Fuqaha, M. Guizani, and J. Qadir, “Trust-Based Cloud Machine Learning Model Selection for Industrial IoT and Smart City Services,” *IEEE IoT-J*, vol. 8, no. 4, 2021.
- [6] F. Yoshioka and I. N., “How do engineers perceive difficulties in engineering of machine-learning systems? - Questionnaire survey,” in *Proc. of IEEE/ACM CESI and SER&IP 2019*, Montreal, Canada, May 2019.
- [7] M. Anisetti, C. A. Ardagna, N. Bena, and E. Damiani, “Rethinking Certification for Trustworthy Machine-Learning-Based Applications,” *IEEE IC*, vol. 27, no. 6, 2023.
- [8] K. Brecker, S. Lins, and A. Sunyaev, “Artificial Intelligence Systems’ Impermanence: A Showstopper for Assessment?” in *Proc. of WITS 2023*, Hyderabad, India, Dec. 2023.
- [9] A. Tahmasbi, E. Jothimurugesan, S. Tirthapura, and P. B. Gibbons, “DriftSurf: Stable-State / Reactive-State Learning under Concept Drift,” in *Proc. of ICML 2021*, Virtual, Jul. 2021.
- [10] W. Zhang and E. Ntoutsis, “FAHT: An Adaptive Fairness-Aware Decision Tree Classifier,” in *Proc. of IJCAI 2019*, Aug. 2019.

- [11] W. Zhang, M. Zhang, J. Zhang, Z. Liu, Z. Chen, J. Wang, E. Raff, and E. Messina, “Flexible and Adaptive Fairness-aware Learning in Non-stationary Data Streams,” in *Proc. of IEEE ICTAI 2020*, Baltimore, MD, USA, Nov. 2020.
- [12] A. Roy, R. M. O. Cruz, R. Sabourin, and G. D. C. Cavalcanti, “A study on combining dynamic selection and data preprocessing for imbalance learning,” *Neurocomputing*, vol. 286, 2018.
- [13] Z.-L. Zhang and Y.-H. Zhu, “Des-Sv: Dynamic Ensemble Selection Based on Shapley Value,” *SSRN preprint SSRN:4608310*, 2023.
- [14] X. Zhu, J. Ren, J. Wang, and J. Li, “Automated machine learning with dynamic ensemble selection,” *Applied Intelligence*, vol. 53, no. 20, 2023.
- [15] P. R. L. Almeida, L. S. Oliveira, A. S. Britto, and R. Sabourin, “Adapting dynamic classifier selection for concept drift,” *Expert Systems with Applications*, vol. 104, 2018.
- [16] R. Chen, Z. Li, J. Li, J. Yan, and C. Wu, “On Collective Robustness of Bagging Against Data Poisoning,” in *Proc. of ICML 2022*, Baltimore, MD, USA, Jul. 2022.
- [17] J. Jia, X. Cao, and N. Z. Gong, “Intrinsic Certified Robustness of Bagging against Data Poisoning Attacks,” in *Proc. of AAAI 2021*, Virtual, Feb. 2021.
- [18] A. Levine and S. Feizi, “Deep Partition Aggregation: Provable Defenses against General Poisoning Attacks,” in *Proc. of ICLR 2021*, Vienna, Austria, May 2021.
- [19] W. Wang, A. Levine, and S. Feizi, “Improved Certified Defenses against Data Poisoning with (Deterministic) Finite Aggregation,” in *Proc. of ICML 2022*, Baltimore, MD, USA, Jul. 2022.
- [20] M. Anisetti, C. A. Ardagna, A. Balestrucci, N. Bena, E. Damiani, and C. Y. Yeun, “On the Robustness of Random Forest Against Untargeted Data Poisoning: An Ensemble-Based Approach,” *IEEE TSUSC*, vol. 8, no. 4, 2023.
- [21] N. Bena, M. Anisetti, G. Gianini, and C. A. Ardagna, “Certifying Accuracy, Privacy, and Robustness of ML-Based Malware Detection,” *SN Computer Science*, vol. 5, no. 6, 2024.
- [22] M. Anisetti, C. A. Ardagna, E. Damiani, and P. G. Panero, “A Methodology for Non-Functional Property Evaluation of Machine Learning Models,” in *Proc. of MEDES 2020*, Abu Dhabi, UAE, November 2020.
- [23] M. Anisetti, C. A. Ardagna, and N. Bena, “Multi-Dimensional Certification of Modern Distributed Systems,” *IEEE TSC*, vol. 16, no. 3, 2023.
- [24] C. A. Ardagna and N. Bena, “Non-Functional Certification of Modern Distributed Systems: A Research Manifesto,” in *Proc. of IEEE SSE 2023*, Chicago, IL, USA, Jul. 2023.
- [25] J. R. Gunasekaran, C. S. Mishra, P. Thinakaran, B. Sharma, M. T. Kandemir, and C. R. Das, “[cocktail: A multidimensional optimization for model serving in cloud],” in *Proc. of USENIX NSDI 2022*, Renton, WA, USA, Apr. 2022.
- [26] F. Doshi-Velez and B. Kim, “Towards A Rigorous Science of Interpretable Machine Learning,” *arXiv preprint arXiv:1702.08608*, 2017.
- [27] M. Anisetti, N. Bena, F. Berto, and G. Jeon, “A DevSecOps-based Assurance Process for Big Data Analytics,” in *Proc. of IEEE ICWS 2022*, Barcelona, Spain, Jul. 2022.
- [28] C. A. Ardagna, R. Asal, E. Damiani, and Q. Vu, “From Security to Assurance in the Cloud: A Survey,” *ACM CSUR*, vol. 48, no. 1, 2015.
- [29] S. Lins, S. Schneider, and A. Sunyaev, “Trust is good, control is better: Creating secure clouds by continuous auditing,” *IEEE TCC*, vol. 6, no. 3, 2018.
- [30] M. Hosseinzadeh, H. K. Hama, M. Y. Ghafour, M. Masdari, O. H. Ahmed, and H. Khezri, “Service Selection Using Multi-criteria Decision Making: A Comprehensive Overview,” *Journal of Network and Systems Management*, vol. 28, no. 4, 2020.
- [31] C. A. Ardagna, R. Asal, E. Damiani, T. Dimitrakos, N. El Ioini, and C. Pahl, “Certification-Based Cloud Adaptation,” *IEEE TSC*, vol. 14, no. 1, 2021.
- [32] V. Iosifidis, T. N. H. Tran, and E. Ntoutsis, “Fairness-Enhancing Interventions in Stream Classification,” in *Proc. of DEXA 2019*, Linz, Austria, Aug. 2019.
- [33] W. Zhang, A. Bifet, X. Zhang, J. C. Weiss, and W. Nejdl, “FARF: A Fair and Adaptive Random Forests Classifier,” in *Proc. of PAKDD 2021*, Virtual, May 2021.
- [34] V. Iosifidis, W. Zhang, and E. Ntoutsis, “Online Fairness-Aware Learning with Imbalanced Data Streams,” *arXiv preprint arXiv:2108.06231*, 2021.

- [35] M. Anisetti, C. A. Ardagna, and N. Bena, "Continuous Certification of Non-Functional Properties Across System Changes," in *Proc. of ICSSOC 2023*, Rome, Italy, Nov.-Dec. 2023.
- [36] R. N. Taylor, N. Medvidović, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, 2009.
- [37] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A Survey on Bias and Fairness in Machine Learning," *ACM CSUR*, vol. 54, no. 6, 2021.
- [38] O. Chapelle and L. Li, "An empirical evaluation of Thompson sampling," in *Proc. of NeurIPS 2011*, Granada, Spain, Dec. 2011.
- [39] S. L. Scott, "Multi-armed bandit experiments in the online service economy," *ASMBL*, vol. 31, no. 1, 2015.
- [40] L. Floridi, M. Holweg, M. Taddeo, J. Amaya Silva, J. Mökander, and Y. Wen, "CapAI - A procedure for conducting conformity assessment of AI systems in line with the EU artificial intelligence act," *SSRN preprint SSRN: 4064091*, 2022.
- [41] S. Maghool, E. Casiraghi, and P. Ceravolo, "Enhancing Fairness and Accuracy in Machine Learning Through Similarity Networks," in *Proc. of CoopIS 2023*, Groningen, The Netherlands, Oct.-Nov. 2023.
- [42] G. Vargas-Solar, C. Ghedira-Guégan, J. A. Espinosa-Oviedo, and J.-L. Zechinelli-Martin, "Embracing Diversity and Inclusion: A Decolonial Approach to Urban Computing," in *Proc. of ACS/IEEE AICCSA 2023*, Giza, Egypt, Dec. 2023.
- [43] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, Sep. 2020.
- [44] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proc. of SciPy 2010*, Austin, TX, USA, Jun.-Jul. 2010.
- [45] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: <https://doi.org/10.5281/zenodo.3509134>
- [46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.
- [47] R. M. O. Cruz, R. Sabourin, G. D. C. Cavalcanti, and T. Ing Ren, "META-DES: A dynamic ensemble selection framework using meta-learning," *Pattern Recognition*, vol. 48, no. 5, 2015.
- [48] R. Mousavi, M. Eftekhari, and F. Rahdari, "Omni-ensemble learning (OEL): utilizing over-bagging, static and dynamic ensemble selection approaches for software defect prediction," *JAIT*, vol. 27, no. 6, 2018.
- [49] P. Pérez-Gállego, A. Castaño, J. Ramón Quevedo, and J. José del Coz, "Dynamic ensemble selection for quantification tasks," *Information Fusion*, vol. 45, 2019.
- [50] M. Badar, W. Nejdil, and M. Fischella, "FAC-fed: Federated adaptation for fairness and concept drift aware stream classification," *Machine Learning*, vol. 112, 2023.
- [51] R. M. Cruz, R. Sabourin, and G. D. Cavalcanti, "Dynamic classifier selection: Recent advances and perspectives," *Information Fusion*, vol. 41, 2018.
- [52] I. Khan, X. Zhang, M. Rehman, and R. Ali, "A Literature Survey and Empirical Study of Meta-Learning for Classifier Selection," *IEEE Access*, vol. 8, 2020.
- [53] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under Concept Drift: A Review," *IEEE TKDE*, vol. 31, no. 12, 2019.



**Claudio A. Ardagna** is Full Professor at the Department of Computer Science, Università degli Studi di Milano, the Director of the CINI National Lab on Data Science, and co-founder of Moon Cloud srl. His research interests are in the area of edge-cloud security and assurance, and data science. He has published more than 140 articles and books. He has been visiting professor at Université Jean Moulin Lyon 3 and visiting researcher at BUPT, Khalifa University, GMU.



**Nicola Bena** is a postdoc at the Department of Computer Science, Università degli Studi di Milano. His research interests are in the area of security of modern distributed systems with particular reference to certification, assurance, and risk management techniques. He has been visiting scholar at Khalifa University and at INSA Lyon.



**Ernesto Damiani** is Full Professor at the Department of Computer Science, Università degli Studi di Milano, where he leads the Secure Service-oriented Architectures Research (SESAR) Laboratory. He is also the Founding Director of the Center for Cyber-Physical Systems, Khalifa University, UAE. He received an Honorary Doctorate from INSA Lyon for his contributions to research and teaching on big data analytics. His research interests include cybersecurity, big data, and cloud/edge processing, and he has published over 680 peer-reviewed articles and books. He is a Distinguished Scientist of ACM and was a recipient of the 2017 Stephen Yau Award.



**Marco Anisetti** is Full Professor at the Department of Computer Science, Università degli Studi di Milano. His research interests are in the area of computational intelligence and its application to the design and evaluation of complex systems. He has been investigating innovative solutions in the area of cloud security assurance evaluation. In this area he defined a new scheme for continuous and incremental cloud security certification, based on distributed assurance evaluation architecture.



**Paolo G. Panero** is a master student at the Department of Computer Science, Università degli Studi di Milano. He is IT officer in a in-house public administration company where he deals with innovation and IT services. His research interests are in the area of Machine Learning with a focus on models evaluation.