Efficient Self-Adaptation through Explanation-Driven White-Box Optimization

FRANCESCO RENATO NEGRI, NICCOLÒ NICOLOSI, and MATTEO CAMILLI, Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milano, Italy
RAFFAELA MIRANDOLA, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

Self-adaptive systems increasingly rely on black-box predictive models, such as Neural Networks, for decision-making and adaptation planning. In this case, adaptation decisions and their potential impact on the surrounding environment are hard to explain. Further, the opaque nature of these models often involves expensive optimization techniques. The computational complexity is a consequence of the inability to directly observe or comprehend the internal mechanisms of the black-box predictive models. This requires iterative methods to explore a possibly large search space and optimize according to many objectives, creating a critical challenge in balancing effectiveness and cost.

In this article, we propose explanation-driven self-adaptation, a novel approach that embeds model-agnostic interpretable machine learning techniques into the feedback loop. In particular, we present XDA-II, an extended version of XDA that integrates multiple explanation sources. This new version combines Partial Dependence Plots and Feature Importance to maintain interpretability while boosting efficiency. Beyond interpretability benefits, explanations become instrumental to guide adaptations through white-box optimization, which is more cost-effective than black-box optimization due to the transparency and predictable mathematical properties of the functions involved. Our empirical evaluation using six study subjects shows that XDA-II achieves more efficient convergence towards optimal adaptation solutions compared to four selected baseline methods including existing black-box methods—random search, NSGA-III, and FITEST—and white-box methods—the predecessor XDA.

CCS Concepts: • **Software and its engineering** → *Software system models*; *Software functional properties*; • **Computer systems organization** → *Self-organizing autonomic computing*;

Additional Key Words and Phrases: Self-adaptation, Model-agnostic explanations, Interpretable Machine Learning, Cyber-physical Systems

This work has received partial funding from the Engineering Secure Systems topic of the Helmholtz Association (HGF), from the pilot program Core Informatics at KIT (KiKIT) of HGFs, and from the Italian Ministry of Education, Universities, and Research (MUR) under the PRIN project SAFEST Award No.: 20224AJBLJ.
Authors' Contact Information: Francesco Renato Negri, Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milano, Italy; e-mail: francescorenato.negri@mail.polimi.it; Niccolò Nicolosi, Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milano, Italy; e-mail: niccolo.nicolosi@mail.polimi.it; Matteo Camilli, Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milano, Italy; e-mail: matteo.camilli@polimi.it; Raffaela Mirandola (corresponding author), Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany; e-mail: raffaela.mirandola@kit.edu.

ACM 1556-4703/2026/6-ART24
https://doi.org/10.1145/3737648

ACM Reference format:
Francesco Renato Negri, Niccolò Nicolosi, Matteo Camilli, and Raffaela Mirandola. 2026. Efficient Self-Adaptation through Explanation-Driven White-Box Optimization. *ACM Trans. Autonom. Adapt. Syst.* 21, 3, Article 24 (June 2026), 31 pages.
https://doi.org/10.1145/3737648

1 Introduction

Today's software-intensive and **Cyber-Physical Systems (CPSs)** increasingly require uninterrupted operation, maintaining their objectives continuously without the need for patches or redeployment. Self-adaptive systems play a key role in enabling real-time responses to unforeseen changes and managing operational uncertainties [1, 2]. The primary engineering strategy for self-adaptation involves adding an adaptation layer to the system architecture. This layer incorporates a feedback control loop operating atop the managed system to drive adaptation actions. These loops typically use analysis, planning, and decision-making techniques that increasingly rely on **Machine Learning (ML)** techniques [3–6].

As outlined by Gheibi et al. [6], ML is typically employed within self-adaptive systems to address specific challenges, such as potential requirement violations or anomalies. Predictive models are commonly built through supervised learning approaches, including **Neural Networks (NNs)** [7], Deep NNs [8], or ensemble models for classification or regression [9]. These ML models are integrated with other software components, data, and artifacts to form the adaptation logic within the control loop. While ML offers significant advantages, its adoption also introduces certain drawbacks, particularly regarding the transparency of adaptation decisions. Furthermore, adaptation mechanisms relying on black-box predictive models often employ expensive iterative methods to explore a possibly large search space and optimize according to many goals to produce adaptation decisions [6, 10].

We argue that enhancing the transparency of these models through explainability can lead to more efficient convergence of optimization methods towards adaptation decisions, resulting in faster adaptation decisions without compromising effectiveness. To the best of our knowledge, the use of explainability as a mechanism to guide adaptation decisions remains largely underexplored.

In this article, we introduce XDA-II,[1] an approach that extends our previous work introduced by Negri et al. [11]. It incorporates model-agnostic interpretable ML [12] into the adaptation feedback loop to drive efficient adaptation decisions while preserving effectiveness.

The main components of XDA-II include (i) the *offline pre-processor* that complements pre-trained classifiers predicting satisfied/unsatisfied requirements with global explanations, showing the relationships between predictions and one or more factors of interest; and (ii) the *online white-box optimizer* that takes advantage of the explanations to steer the search for possible adaptation options. Compared to its predecessor, XDA-II combines multiple explanation sources: **Partial Dependence Plots (PDPs)** [12] and **Feature Importance (FI)** [13] to maintain interpretability while boosting efficiency. The integration and combination of these explanations allow us to achieve more efficient convergence towards optimal adaptation solutions leveraging white-box optimization mechanisms. XDA-II supports multiple requirements simultaneously and is completely automated.

It is worth noting that the primary goal of XDA-II is to enhance transparency for the purpose of improving the efficiency and cost-effectiveness of automated decision-making, rather than

[1]XDA stands for eXplanation-Driven self-Adaptation.

explicitly aiming to make decisions more understandable to humans. We acknowledge that increased interpretability can also improve human understandability as a beneficial side effect.[2]

We conducted an empirical evaluation to study the cost-effectiveness and scalability of XDA-II using three existing case studies: a self-adaptive search and rescue robotic system for emergency circumstances [14], an autonomous vehicle equipped with an automated emergency braking component [15], and an autonomous team of **Unmanned Aerial Vehicles (UAVs)** that carry out a surveying mission in a hostile environment [16]. We used two different versions of each case study, for a total of six study subjects having increasing complexity in terms of a number of requirements and factors of interest in the operating conditions (environment and configuration of the system). Compared to our prior work [11], the main contributions of XDA-II can be summarized as follows:

- —extended methodology that includes the combination of heterogeneous interpretable ML techniques to boost efficiency while maintaining effective adaptation decisions;
- —redesigned empirical evaluation that includes (i) additional **Research Questions (RQs)**, (ii) four additional study subjects, (iii) evaluation of the statistical significance of results, and (iv) a quantitative comparison of XDA-II with selected baseline methods including the predecessor XDA and other mainstream approaches based on black-box optimization; and
- —a publicly available replication package, including sources of XDA-II and instructions to replicate our experiments.

The remainder of the article is organized as follows. Section 2 presents the problem addressed in the article together with the description of a running example in the robotics domain. Section 3 introduces background notions we use in the rest of the article. The XDA-II approach is illustrated in Section 4, and the empirical evaluation is described in Section 5. In Section 6, we discuss threats to validity as well as advantages and disadvantages of XDA-II. Section 7 discusses related work, and Section 8 concludes the article.

2 Problem Statement

We provide a problem description regarding opaque decisions guided by black-box predictive models in self-adaptive systems. As a running example, we use a search-and-rescue robotic system [14] to illustrate our main arguments.

The primary objective of the **Rescue Robot (RR)** in our example, as outlined by Esfahani et al. [14], is to locate individuals in emergency scenarios such as fires, hurricanes, or earthquakes. The robot (i.e., the managed system) is equipped with abstracted physical interfaces (sensors and actuators) and essential functions for performing rescue tasks, including navigation, obstacle detection, and human detection with avoidance capabilities.

The robot's main components can be reconfigured at runtime to adapt its operating mode in response to environmental changes (e.g., available bandwidth) and resource variations (e.g., estimated battery life). These adaptations are managed by an **Monitor, Analyse, Plan, Execute and Knowledge (MAPE-K)** feedback control loop, which identifies the need for adjustments, selects appropriate strategies, and implements them to meet adaptation goals, such as maintaining dependability under dynamic conditions.

In critical settings such as our RR example, requirements often emphasize safety, such as:

—R_1 "*The RR shall not misclassify obstacles in more than 25% of cases*"; or

[2]We refer the reader to existing literature that demonstrates the general effectiveness of FI [13] and PDPs [12] in enhancing human understanding of opaque predictive models. While improving human interpretability is a recognized benefit of these techniques, it falls outside the primary scope of our work.

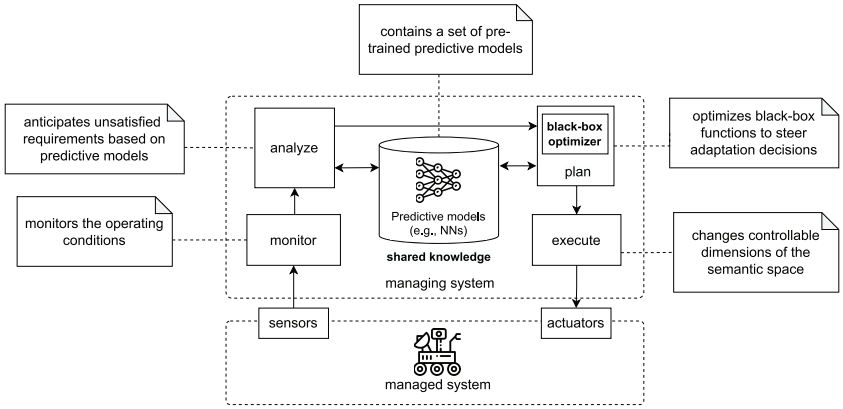


Fig. 1. Schema of our illustrative example without XDA.

Table 1. Semantic Space of the Search and RR

Variable	Dimensions	Type	Domain
Cruise speed	CF	Continuous	[0, 5] m/s
Camera quality	CF	Categorical	{*low*, *mid*, *high*}
Control responsiveness	CF	Continuous	[10, 50] Mbit/s
Illuminance	CF	Continuous	[40, 120,000] lux
Power level	OF	Integer	[0, 100] %
Smoke intensity	OF	Categorical	{*none*, *thin*, *thick*}
Obstacle size	OF	Continuous	[0, 2] m^3
Obstacle distance	OF	Continuous	[0, 10] m
Firm obstacle	OF	Boolean	Yes/no

— R_2 "*While moving, the RR shall maintain a protective distance from humans in at least 90% of the cases.*"

As illustrated in Figure 1, the adaptation decisions actuated by the control loop can be informed by one or more pre-trained classifiers (e.g., NN models) maintained by the knowledge component. These classifiers predict when dependability requirements might not be met and guide configuration changes to improve compliance based on the operating conditions [10]. These conditions include both **Controllable Features (CFs)**, such as cruise speed, and **Observable Features (OFs)**, like power level. Table 1 outlines a selected set of CFs and OFs that together form the *semantic space* of the RR system [17, 18].

The decision-making process within the control loop is influenced by the black-box nature of the embedded NNs. In this context, adaptation decisions are typically the outcome of black-box optimization processes that maximize the likelihood of satisfying the requirements (adaptation goals). Optimizing black-box functions with extensive search spaces presents a significant challenge in terms of costs. The complexity arises from the inability to directly observe or comprehend the internal mechanisms of the NNs, which requires iterative methods (e.g., evolutionary algorithms) to explore a possibly large semantic space and optimize according to many goals. Balancing effectiveness against cost is critical, as these techniques frequently encounter the "curse of dimensionality," which further amplifies the cost of runtime adaptation decisions [19]. This issue is

even exacerbated by the fact that autonomous systems often need to satisfy *many*[3] requirements simultaneously. While the RR example above simplifies the scenario by mentioning only two requirements, real-world applications typically involve more constraints (e.g., safety regulations, routing rules, collision avoidance).

We address these challenges by introducing a novel adaptation method called **eXplanation-Driven self-Adaptation (XDA)**. The initial version of XDA, presented by Negri et al. [11], leverages interpretable ML to enhance the transparency of predictive models embedded within the control loop. Specifically, XDA uses PDPs [12] to generate explanations to make the models more interpretable (white-box), which in turn improves guidance for adaptation decisions. In this article, we enhance XDA by integrating multiple explanation sources in our extended version, XDA-II. This new version combines PDPs and FI to maintain interpretability while boosting efficiency. Beyond interpretability benefits, white-box optimization can often be more cost-effective than black-box optimization due to the transparency and predictable mathematical properties of the functions involved. By integrating and combining these explanations, XDA-II enhances decision-making with more efficient convergence toward optimal adaptation solutions. The selection of these two explanation techniques is grounded in their complementary strengths. FI highlights which CFs have the greatest impact on predictions, allowing the system to focus optimization efforts where they matter most—thus reducing unnecessary exploration in the semantic space. PDPs yield insights into how changes in individual CFs affect the outcome, helping to identify effective directions for adaptation. Together, these explanations are used by XDA-II to realize strategic prioritization (via FI) and effective optimization directions (via PDPs).

Example 1 (Enhanced Decision-Making). Let us consider the RR example. When navigating a hazardous environment, understanding the relative importance of different CFs, such as cruise speed or camera quality, can significantly enhance decision-making. By leveraging FI, the adaptation mechanism can prioritize the most impactful adjustments to mitigate potential requirement violations. Additionally, PDPs provide guidance into promising "optimization directions" by illustrating how modifications to key CFs influence system safety. For instance, if a PDP reveals that reducing lower cruise speed has an uneven effect on safety under high smoke intensity (low visibility), the system may adapt by rerouting the robot to avoid the smoky area rather than reducing speed.

3 Background

We introduce background concepts to make the article self-contained. In particular, we introduce the notions of *supervised* and then *interpretable* ML, focusing on model-agnostic methods. We refer the reader to the provided bibliography for a comprehensive treatment of these topics.

3.1 Supervised ML

In the field of ML, *classification* [21] refers to a predictive modeling problem where the class label y_i is anticipated for a specific input data point x. Classification models (either binary or multiclass) are built by using supervised learning techniques to create a concise representation of the distribution of class labels in terms of quantifiable properties, known as features (or explanatory variables). Thus, a data point x is a vector that contains a value x_j for each feature j. A supervised learning algorithm that implements classification is referred to as *classifier*. Supervised learning uses a *training set* that includes pre-labeled data points $\langle x, y_i \rangle$ to "learn" the desired classification function $\hat{f}$. The classification function applied to a data point $\hat{f}(x)$ is referred to as *prediction*. Class probability

[3]We use the term "many" to refer to a typical range of 4–15, as commonly considered in the context of many-objective optimization [20].

estimation summarizes the likelihood of a data point belonging to a given class label. We denote it as $\mathcal{P}_M(y = y_i|x)$, where M is the classifier, x is a data point, y_i is a class label.

There exist several popular classifiers (either binary or multiclass) in supervised ML, including, for instance, NNs [21]. Some predictive models are designed to have a clear and simple structure, and their predictions are inherently explained (e.g., Linear Regression, Decision Trees). More complex models (e.g., NNs, **Random Forests (RFs)**) that do not explain their predictions are referred to as *black-box* models.

3.2 Interpretable ML

Interpretable ML [12] refers to the extraction of relevant knowledge from an ML model concerning existing relations contained in data or learned by the model itself. In this sense, interpretability is the ability of a model to be understood and explained by humans. According to the terminology introduced by Miller [22], we use the terms *interpretable* and *explainable* interchangeably.

The scope of interpretability is either *global* (i.e., holistic model interpretability) or *local* (i.e., interpretability for a single prediction). Global explanations describe the average behavior of a given model. They give a holistic view of the distribution of the target outcome (e.g., class labels) based on the features. This helps in understanding which features are important and what kind of interactions between them exist.

3.2.1 PDP. PDP [12] is a global model-agnostic method that shows the marginal effect that selected features have on the predicted outcome of a model. A PDP shows the relationship between the predictions and one or more features (e.g., linear, monotonic, or more complex). The PDP function is defined as follows:

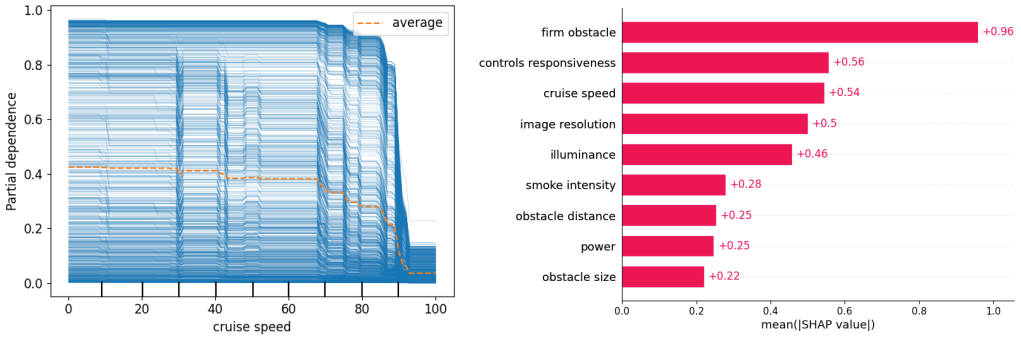
$$\hat{f}_S(x_S) = E_{X_C}[\hat{f}(x_S, X_C)] = \int \hat{f}(x_S, X_C)\, d\mathbb{P}(X_C), \tag{1}$$

where x_S are the features for which we want to know the effect on the prediction (usually one or two features), and X_C are the other features treated here as random variables. PDP marginalizes the model output over the distribution of features X_C to show the relationship between x_S and the predictions. For classifiers, PDP calculates the probability for a certain class label given different values for feature(s) x_S.

Example 2 (PDP Explanation). Figure 2(a) shows an example of PDP in our illustrative RR system. The plot shows both *average* (dashed line) and *individual* conditional expectations (continuous lines) by taking into account a selected CF. The individual conditional expectation is the dependence of the prediction on the selected CF for each sample separately with one line per sample. In this example, the plot shows the dependence between the probability of satisfying R_1 and the cruise speed of the robot. In general, we can see a sharp decrease when the speed is above 80% (4 m/s). Some of the lines are flat (and close to 0.0), no matter what is the speed, while some other lines reach higher values (close to 1.0).

3.2.2 FI. FI aims to determine the influence of each feature, that is, it allows us to understand which features drive predictions and by how much. Global explanations in this area give an overview of FI across a given dataset, showing which features are influential on average across all predictions. In the following, we briefly introduce two popular model-agnostic methods: **Permutation FI (PFI)** [13] and **SHapley Additive exPlanations FI (SHAP-FI)** [23].

PFI assesses the importance of a feature by calculating the increase in the prediction error after permuting the feature. Indeed, a feature is "more important" if shuffling its values increases the model error, while a feature is "less important" if shuffling its values leaves the model error



(a) Selected example of PDP

(b) Selected example of SHAP-FI

Fig. 2. Selected explanations extracted from the RR illustrative example.

unchanged. In this case, the model ignores the feature for the prediction. For each feature j, PFI can be calculated as the quotient e_{perm}/e_{orig}, with e_{orig} and e_{perm} prediction errors before and after permuting j, respectively. Features can be sorted by descending PFI to rank them by importance.

SHAP is a method based on cooperative game theory which makes use of the so-called Shapley values to calculate the contribution of each feature to predictions [23]. According to Lundberg et al. [23], Shapley values assess the impact of including each feature by calculating its marginal contribution across all possible subsets of features, capturing how much each feature adds to corresponding predictions when present versus absent. SHAP-FI derives global FI by averaging the absolute Shapley values across all predictions in a given dataset. While PFI is based on the decrease in model performance, SHAP-FI is based on the magnitude of feature contribution.

Example 3 (SHAP-FI Explanation). Figure 2(b) shows an example of SHAP-FI in our illustrative RR system for the classifier predicting satisfaction of requirement R_1. A higher mean SHAP value for a feature indicates that, on average, the feature has a larger impact on the classification, whether positively or negatively, relative to other features. Features are ranked from bottom to top based on their increasing mean SHAP values. Here, "firm obstacle" (the top-ranked feature) has an impact approximately 40% greater than "control responsiveness" (the second-ranked feature) and around 73% higher than "obstacle size" (the lowest-ranked feature).

4 The XDA-II Approach

This section presents XDA-II which addresses the problem introduced in Section 2. We introduce an overview of the approach and a detailed description of the main components implementing the two stages of XDA-II: (1) offline pre-processing and (2) online white-box optimization.

4.1 Overview

Figure 3 illustrates a high-level overview of XDA-II and its integration into a MAPE-K control loop. As described in Section 2, we assume that the knowledge maintains a set of pre-trained binary classifiers $M = \{M_1, ..., M_n\}$, one for each requirement R_i. The *monitor* component periodically samples the current operating condition of the managing system. The operating condition is an assignment to controllable features $CF = \hat{CF}$ and observable features $OF = \hat{OF}$ fed into the *analyze* component. Thus, the *analyze* takes as input the pair $(\hat{CF}, \hat{OF})$ and uses each classifier M_i to predict the satisfaction of the corresponding requirement R_i: class label *true* (R_i satisfied) or class label *false* (R_i unsatisfied). If there exists at least one requirement whose predicted class label is *false*, the

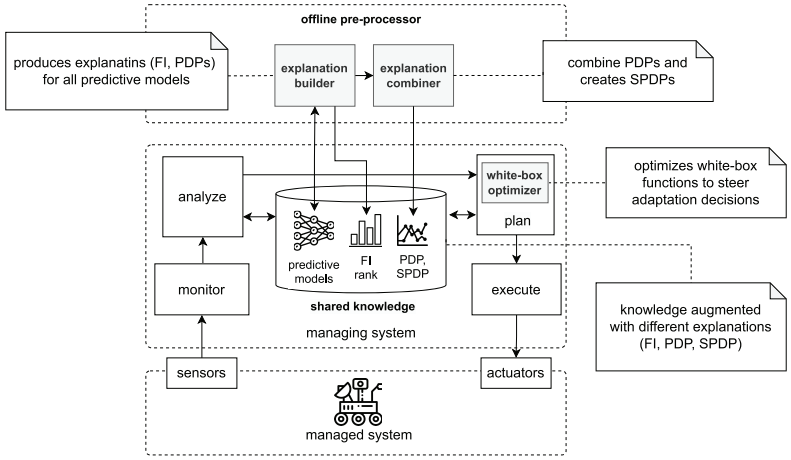


Fig. 3. Main components of XDA-II embedded in an MAPE-K loop.

analyze triggers the *plan* component that searches for one or more new assignments to CFs that increase the probability of satisfying all the requirements. Such assignments represent alternative adaptation options among which the final adaptation decision can be made according to a given policy (e.g., highest class probability estimate). The final decision can then be enacted by the *execute* component in charge of modifying the configuration of the managed system.

As illustrated in Figure 3, XDA-II relies on three main components executed in separate phases of the system lifecycle. The explanation builder and the explanation combiner belong to the *offline pre-processor* in charge of augmenting the knowledge with a set of explanations. The pre-processor works offline, before deploying the system into its intended environment where it will run and serve its purpose. The *online white-box optimizer* uses the augmented knowledge to explore adaptation options based on current operating conditions. Specifically, leveraging explanations that make predictive models interpretable (white-box), it enhances guidance for adaptation decisions. The white-box optimizer is part of the plan and, as such, runs with the other components of the MAPE-K control loop after the deployment of the whole system in its execution environment.

Both pre-processing and white-box optimization of XDA-II are fully automated and supported by a software implementation we used to carry out an empirical evaluation.

Notice that our illustrative example states that the knowledge includes black-box classifiers, such as NNs (see Section 2). These classifiers are generic and not limited to NNs. For instance, they could include RFs or Gradient-Boosting Machines. Indeed, XDA-II employs model-agnostic interpretable ML techniques, allowing it to function independently of the specific family of predictive models. This flexibility ensures consistent interpretability across a range of models.

4.2 Offline Pre-Processor

Figure 3 shows that the offline pre-processor includes two components: an explanation *builder* and an explanation *combiner*. The builder directly interacts with the set of classifiers M in the knowledge and runs global interpretable ML using PDPs and FI. PDPs offer peculiar advantages in understanding the relationship between a given feature and the target prediction of the black-box classifiers contained in the knowledge. PDPs specifically demonstrate how individual features influence model predictions. For instance, Figure 2(a) shows that according to both average and individual conditional expectation curves, the cruise speed is not an influential feature in the range $[0, 80]\%$, while there is a steeper slope that indicates a stronger impact after 80%. FI, on the

Algorithm 1: Offline Pre-Processing

Input: Set of classifiers $M = \{M_1, \ldots, M_n\}$, Set of controllable features $CF = \{CF_1, \ldots, CF_m\}$
Output: Set of PDPs $PDP = \{PDP_{ij} \forall i = 1..n, j = 1..m\}$, set of FI ranks $FI = \{FI_i \forall i = 1..n\}$, set of m summary PDPs $SPDP = \{SPDP_1, \ldots, SPDP_m\}$

1 Set $FI \leftarrow \emptyset$
2 Set $PDP \leftarrow \emptyset$
3 Set $SPDP \leftarrow \emptyset$
/* Build step: creation of PDPs and FIs */
4 **for** $i \leftarrow 1$ **to** n **do**
5 **for** $j \leftarrow 1$ **to** m **do**
6 $PDP_{ij} \leftarrow buildPDP(M_i, CF_j)$
7 $PDP \leftarrow updateSet(PDP, PDP_{ij})$
8 $FI_i \leftarrow buildFI(M_i, CF)$
9 $FI \leftarrow updateSet(FI, FI_i)$
10 **end**
11 **end**
/* Combination step: creation of SPDPs */
12 **for** $j \leftarrow 1$ **to** m **do**
13 $score_j \leftarrow buildScoreFunction(PDP_{1j}, \ldots, PDP_{nj})$
14 $SPDP \leftarrow updateSet(SPDP, score_j)$
15 **end**

other hand, highlights which features are most influential on average across all predictions. For instance, Figure 2(b) indicates that cruise speed and control responsiveness have comparable levels of importance and are, overall, the two CFs with the greatest impact on predictions. Intuitively, the combination of FI and PDPs can guide the selection of new assignments to CFs. FI helps identify which features are most impactful and should be prioritized for adjustment, while the corresponding PDP offers guidance on the direction and extent of these modifications.

Algorithm 1 shows the high-level procedure implemented by the pre-processor component. Given the set of classifiers M and the set of controllable features CF, the algorithm builds a PDP_{ij} for each pair M_i, CF_j (lines 6–7). Each PDP_{ij} yields the relationship between the expected satisfaction of R_i and one selected controllable feature CF_j. The PDP includes both average behavior (dashed line in Figure 2(a)) and individual conditional expectation (continuous lines in Figure 2(a)). Let $PDP_{ij}(s)$ be the function that represents the individual conditional expectation for a given sample s in the training set. We denote $PDP_{ij}(s, x)$ as the punctual expectation for R_i when $CF_j = x$, holding the other features at the values in sample s. In addition to PDPs, the builder creates all FI ranks for each classifier in M, considering the whole set of controllable features CF (lines 8–9).

After creating all the PDPs, the combiner receives all PDPs and creates a score function for each CF_j, considering all models (line 13). The score function for each sample is essentially the product of all individual conditional expectation curves across all models, as follows:

$$score(j, s, x) = \prod_i PDP_{i,j}(s, x). \tag{2}$$

According to Equation (2), the score is high if all values $PDP_{i,j}(s, x)$ are high, meaning that x is a "good" assignment for CF_j since it would lead high expected probability for the satisfaction of all requirements R_i. On the contrary, the score is low if all values $PDP_{i,j}(s, x)$ are low. The whole score is zero if there is at least one requirement R_i such that $PDP_{i,j}(s, x) = 0$, even with high expected probability for the other requirements.[4] It is important to note that the score does not necessarily indicate the probability of meeting all requirements, as the fulfillment of these requirements may not be independent events. For example, in RR (see Section 2), the robot's ability to maintain a safe human–robot distance (relevant to requirement R_2) could be influenced by errors in its visual

[4]In this case, we chose to penalize the assignment $CF_j = x$ as it would inevitably result in violating one or more requirements.

Algorithm 2: Online White-Box Optimization (Climbing)

Input: Set of CFs $CF = \{CF_1, \ldots, CF_m\}$ and corresponding values $\hat{CF} = \{\hat{CF}_1, \ldots, \hat{CF}_m\}$, set of OFs $OF = \{OF_1, \ldots, OF_l\}$ and corresponding values $\hat{OF} = \{\hat{OF}_1, \ldots, \hat{OF}_l\}$, set of stacks of CFs (one stack for each classifier) $FI = \{FI_1, \ldots, FI_n\}$, set of summary PDPs $SPDP = \{SPDP_1, \ldots, SPDP_m\}$ and corresponding PDPs $\{PDP_{1,j}, \ldots, SPDP_{n,j}\}$ for each $SPDP_j$, number of neighbors k, training set D used to train the classifiers.
Output: Set of changed operating conditions S

```
1  S ← ∅
2  U ← neighborhood(ĈF, ÔF, D, k)
3  foreach (ĈF′, ÔF′) ∈ U do
4      i ← 0
5      min ← 1.0
6      F ← {1...n}
7      while F ≠ ∅ do
8          foreach j ∈ F do
9              k ← argmin_x(PDP_{x,j}(ĈF′_j))
10             if PDP_{k,j}(ĈF′_j)) ≤ min then
11                 i ← k
12                 min ← PDP_{i,j}(ĈF′_j))
13             end
14         end
15         CF_h ← pop(FI_i)
16         ĈF′_h ← argmax_x(SPDP_h(x))
17         ĈF″_h ← ĈF′_h
18         {(ĈF′, ÔF′)} ← neighborhood(ĈF′, ÔF, D, 1)
19         remove(F, h)
20     end
21     S ← updateSet(S, (ĈF″, ÔF))
22 end
```

perception subsystem (relevant to requirement R_1). As a consequence, in this scenario, meeting requirement R_2 is dependent on the fulfillment of R_1.

The outcome of Algorithm 1 is composed of the set *PDP* and *FI* that include all explanations in the form of PDPs and FI ranks, respectively, as well as the set of score functions, henceforth denoted by *SPDP* (i.e., Score PDP). These artifacts are stored in the knowledge component since they represent the input for the white-box optimizer.

4.3 Online White-Box Optimizer

The online white-box optimizer is integrated into the plan component of the MAPE-K control loop. As outlined in Section 4.1, we assume that the analyze component triggers the plan component whenever the operating condition leads to negative predictions—specifically, when the class label predicted by the black-box classifiers for one or more requirements is *false*. Each time the plan component is activated, the white-box optimizer leverages information from previously generated explanations (FI, PDPs, and SPDPs) to produce a new set of assignments for CFs. Starting from the current operating condition $(\hat{CF}, \hat{OF})$, the optimizer prioritizes CFs with higher impact, as determined by the FI rankings. Guided by the PDPs, it then constructs a set $\{CF = \hat{CF}^*\}$ such that $(\hat{CF}^*, \hat{OF})$ results in a class label of *true* with an estimated class probability exceeding a specified threshold P for all requirements.

The procedure implemented by the white-box optimizer includes two sequential steps: *climbing* and *descending* listed in Algorithm 2 and Algorithm 3, respectively. The climbing step performs coarse-grained changes to CFs according to both (summary) PDPs and FIs to maximize the expected probability of satisfying all requirements. The descending step refines the climbing outcome by

Algorithm 3: Online White-Box Optimization (Descending)

Input: Set of classifiers $\{M_1, \ldots, M_n\}$ associated with requirements, probability threshold P, set of CFs $CF = \{CF_1, \ldots, CF_m\}$ and corresponding values $\hat{CF} = \{\hat{CF}_1, \ldots, \hat{CF}_m\}$, set of OFs $OF = \{OF_1, \ldots, OF_l\}$ and corresponding values $\hat{OF} = \{\hat{OF}_1, \ldots, \hat{OF}_l\}$, set of ideal values for CFs $I = \{I_j\}$, ordered set of CFs for each classifier $FI = \{FI_1, \ldots, FI_n\}$, set of summary PDPs $SPDP = \{SPDP_1, \ldots, SPDP_m\}$, training set D of the classifiers in M, descent step Δ, set of operating condition S computed by the climbing step.

Output: Set of adaptations $\{\hat{CF}^* = \{\hat{CF}^*_1, \ldots, \hat{CF}^*_m\}\}$

```
1  foreach (ĈF*, ÔF) ∈ S do
2      foreach i ∈ 1...m do
3          while FI_i ≠ ∅ do
4              CF_I ← pop(reverse(FI_i))
5              {(ĈF*, _)} ← neighborood(ĈF*, ÔF, D, 1)
6              prev ← ĈF*_I
7              ĈF*_I ← move(ĈF*_I, I_I, Δ)
8              while prev ≠ ĈF*_I ∧ P_{M_j}(y = true|(ĈF*, ÔF)) ≥ P ∀j do
9                  prev ← ĈF*_I
10                 ĈF*_I ← move(ĈF*_I, I_I, Δ)
11             end
12         end
13     end
14 end
```

following the score function curves in the opposite direction. In this case, the procedure slowly decreases the expected probability of success with a relatively low rate of descent to improve the value of CFs (if possible) while keeping the probability greater than the threshold P for all requirements.

4.3.1 Climbing. The input of the climbing procedure in Algorithm 2 includes the set of controllable features CF and observable features OF, as well as the corresponding assignment $(\hat{CF}, \hat{OF})$ that is the data point which represents the current operating condition. Starting from this point, the procedure generates a set of known *neighbors*, denoted as *neighborhood* U (line 2). This set is composed of the k nearest samples extracted from the training set D, where k and D are inputs to the procedure. The goal is to make deductions about the current operating condition (potentially unseen data point) by leveraging past, similar instances previously processed and stored within the knowledge component.

The procedure retrieves the neighborhood measuring the distance between data points belonging to the semantic space. Since each point is a heterogeneous vector composed of numerical and categorical values, we adopt a *heterogeneous distance* metric,[5] as suggested by Wilson and Martinez [24].

For each neighbor $(\hat{CF}', \hat{OF}')$, it inspects the PDPs, it finds the requirement R_i that reaches the lowest value (lines 8–14). Then, the procedure retrieves the most important (top-ranked) feature CF_h according to the corresponding ranking FI_i (line 15) to change its value maximizing the expected score given by the function $SPDP_h$ (line 22). Since this typically leads to a new potentially unexplored assignment, the procedure moves to the nearest neighbor (line 18). Then it repeats the same process by excluding the modified feature CF_h (line 19) until all the features have been processed. At this point, the new assignment to controllable features $\hat{CF}''$ becomes a possible adaptation option by including the whole operating condition $(\hat{CF}'', \hat{OF})$ into the set S (line 21).

Example 4 (Climbing). Consider the example illustrated in Figure 4. For simplicity, the example shows the application of the climbing step limited to a subset of the CFs available for the RR system: cruise speed and light intensity (or illuminance) of a lamp installed onto the robot and used to light up the field of view of the vision sensors. The flow of operations is as follows. According to PDPs,

[5]We define the distance as the average of the normalized *Hamming* distance of categorical variables and the normalized *Manhattan* distance of numeric variables, where the latter are normalized by their maximum range of values.

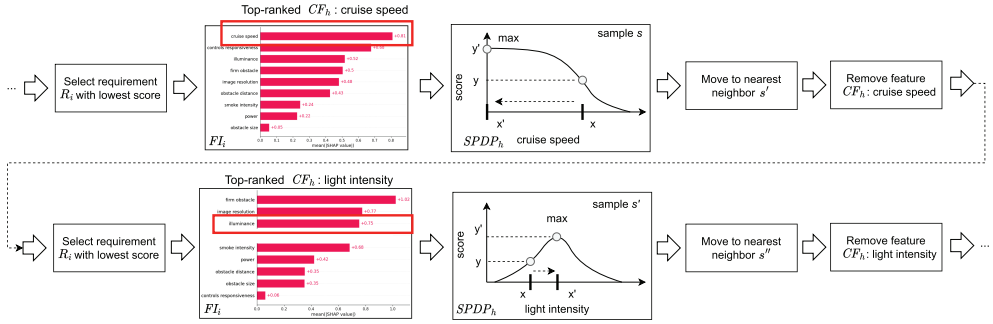


Fig. 4. Example of climbing step limited to two CFs.

the procedure selects the requirement R_i that currently has the lowest score. Then, the procedure selects the most important CF for R_i, that is, cruise speed as indicated by the pre-computed FI FI_i. Thus, the procedure changes the cruise speed of the current operating mode s from value x to x' to reach the maximum score value y' according to the score function. After applying this change, the procedure moves to the nearest neighbor s', then it repeats the same process with the next feature, that is, it selects the requirement having the lowest score, then it modifies the top-ranked feature, light intensity, from x to x' to maximize the score. The new cruise speed and light intensity compose a possible adaptation option further fine-tuned through the descending step.

Compared to its predecessor, XDA-II optimizes the selection process for CFs by leveraging pre-calculated FI rankings for each requirement, obtained during offline pre-processing. In this case, feature selection operates in constant time $\Theta(1)$. In contrast, without this pre-computed information, XDA must generate these rankings dynamically in each iteration of the climbing step for each neighbor. According to Negri et al. [11], the time complexity of this extra step is $\Theta(m!)$ as it iterates over the CFs and, by inspecting the score functions, it finds the feature that reaches the highest score. The corresponding feature value yields the identification of the closest neighbor. Then the same process is repeated by excluding the modified feature until all the features have been processed.

4.3.2 Descending. The descending step refines the set of possible adaptation options generated by the climbing step. This refinement aims at improving trivial adaptations by carefully moving the value of CFs toward ideal operating conditions while maintaining the expected probability above the given threshold P for all requirements.

Example 5 (Trivial Adaptation). Let us examine the safety requirement R_2 for the RR system, that is, "while moving, the robot shall maintain a protective distance from humans in at least 90% of the cases." A straightforward adaptation to meet this requirement might be to reduce the cruise speed to zero, which would effectively stop the robot's movement and satisfy the requirement trivially, reducing the usefulness of the system. Indeed, movement is crucial for the robot to fulfill mission goals quickly reaching critical locations within the search area. Therefore, the ideal condition would involve maintaining a relatively high (even the highest) cruise speed throughout its operation.

The descending step works under the assumption that engineers, leveraging their domain knowledge, define the *ideal* operating conditions before deploying the system, as illustrated by the following example.

Example 6 (Ideal Operating Condition). Let us consider again the RR system. In this scenario, the robot should ideally move as fast as possible to target locations. Thus, a cruise speed of 100%, which corresponds to the highest value within its domain (i.e., 5 m/s), would represent, in this case,

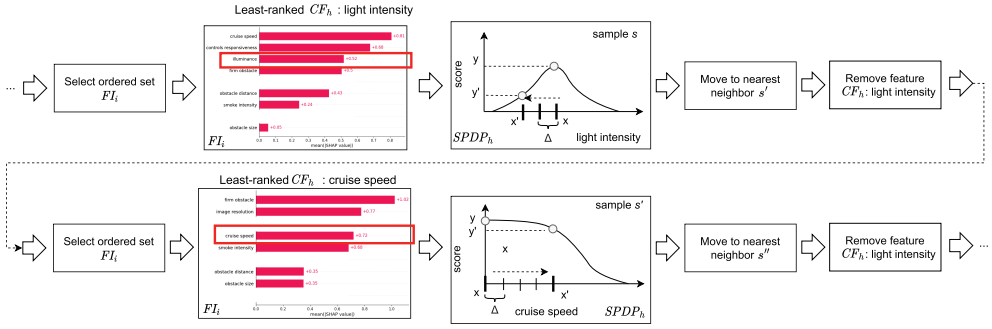


Fig. 5. Example of descending step limited to two CFs.

the ideal operating condition for this feature. Concerning camera quality, the robot should reduce it, especially when high resolution is unnecessary to reduce energy consumption. In this case, the ideal operating condition for the feature is 0%, the lowest level in its range (i.e., *low* quality).

According to Algorithm 3, the set of ideal values $I = \{I_1, ..., I_m\}$, one value for each CF, is an input required by the descending step. In scenarios where engineers cannot define the ideal point I_j, the descending step is not executed for the corresponding controllable feature CF_j.

The descending step considers each adaptation option at a time (line 1). The process sequentially iterates through the ordered sets of CFs in FI. For each FI_i, the order is reversed (line 2) to evaluate CFs from the lowest-ranked to the highest-ranked. Then, the procedure modifies the value of the selected CF depending on the descending direction identified by the procedure *move* (line 7). Identifying the descending direction involves comparing $\hat{CF}_j$ and the ideal value I_j. Namely, if $\hat{CF}_j < I_j$, the procedure tries to increase the feature value. The procedure tries to decrease the feature value if $\hat{CF}_j > I_j$. It is worth noting that the order of CFs is reversed since we favor features that have little impact on the satisfaction of requirements (i.e., they yield gradual descent rates). Intuitively, this gradual descent offers a better chance to approach proximity to the ideal value while minimally reducing the probability of fulfilling requirements. Consequently, the procedure sequentially processes each CF by adjusting its value in the optimization direction by a fixed constant Δ (line 7) provided as input to the descending procedure. Such fine-tuning of each feature through incremental adjustments continues until either no further changes are possible or the probability estimation for class *true* is below the threshold P for at least one requirement (lines 8–10).

Example 7 (Descending). Consider the changes applied to two CFs illustrated in Figure 5. We limit the example again to two selected CFs of the RR: cruise speed and light intensity. As anticipated above, the ideal points are cruise speed at 100% and light intensity at 0%. This implies that the descending directions are left-to-right and right-to-left for cruise speed and light intensity, respectively. In this scenario, light intensity is the least-ranked CF according to the selected FI_i. Thus, starting from the current sample s, the procedure applies incremental adjustments to the current light intensity value, decreasing the feature value by Δ. This modification shifts the feature from its initial value x to a new value x', aiming for closer proximity to the ideal value while ensuring a high probability of meeting the requirements. After applying this change, the procedure moves to the closest neighbor s' and removes the considered feature light intensity. Then it repeats the same process with the next (least-ranked) feature. In this example, it adjusts the cruise speed from x to x' through incremental Δ shifts in the corresponding descending direction.

Compared to the predecessor XDA, the new version XDA-II improves efficiency by relying on pre-calculated information given by FI. Indeed, XDA iteratively recalculates rankings through each

adaptation option, by evaluating each CF based on a metric termed *goodness of slope*. The complexity of this extra step is linear in the number of CFs $\Theta(m)$. However, calculating the goodness of slope for each feature is expensive as it requires deriving and comparing the gradients of all score curves at specific operating conditions. XDA-II significantly reduces this complexity to constant time $\Theta(1)$ by leveraging pre-computed rankings.

5 Evaluation

This section reports on the empirical evaluation of XDA-II[6] using six evaluation subjects having increasing complexity. We answer the following RQs:

- *RQ1*: What is the effectiveness of XDA-II compared to selected baseline methods?
- *RQ2*: What is the quality of the adaptation decisions computed by XDA-II compared to selected baseline methods?
- *RQ3*: What is the cost of the online stage of XDA-II compared to selected baseline methods?
- *RQ4*: What is the cost overhead introduced by the offline stage of XDA-II?

To answer RQ1, we compare the results obtained by using different adaptation planning strategies: our approach XDA-II and selected baselines including **Random Search (RS)**, the predecessor XDA [11], and methods based on existing metaheuristic optimizing search algorithms NSGA-III [20], and FITEST [25]. All selected baseline methods are many-objective as XDA-II. Specifically, we selected representative black-box approaches that do not incorporate explainability but rely on predictions made by black-box models only (NSGA-III and FITEST) and white-box approaches that leverage explainability (the previous version of XDA). We answer RQ1 by comparing two selected effectiveness metrics: (i) *satisfaction likelihood* defined as the predicted probability of satisfying requirements given the adaptation decisions; and (ii) *success rate* defined as the actual satisfaction of requirements observed after the actuation of the adaptation decisions. We answer RQ2 by measuring and comparing the *quality score* of the adaptation decisions computed as the magnitude of the difference between the actuated and target (ideal) operating conditions. We answer RQ3 by comparing the cost (in execution time) of running the different planning strategies. We answer RQ4 by measuring the cost overhead (in execution time) introduced by the offline stage of XDA-II.

5.1 Design of the Evaluation

5.1.1 Evaluation Subjects. We designed and carried out an experimental campaign to evaluate XDA-II and the other baseline approaches using the same set of evaluation subjects: three open source CPS benchmarks, each one in two versions, resulting in a total of six subjects. Table 2 lists the subjects, their main characteristics, and the studies describing them in more detail. We selected existing benchmarks having non-trivial complexity according to different factors, including varying levels of structural complexity, defined by the size of their semantic space (i.e., the number of OFs and CFs) and the number of requirements to be satisfied. To evaluate XDA-II under different conditions, each study subject is provided in two variants: one with half and one with double the number of CFs. This setup enables a controlled investigation into how adaptation strategies perform across systems of increasing complexity.

The ADAS benchmark [15] is an example of an autonomous vehicle equipped with an automated emergency braking component. The vehicle drives down an urban street while pedestrians may cross. Runtime adaptation may be applied to enforce safety requirements, such as the ability to drive within the boundaries of the lane and the ability to maintain a safe distance from the vehicle

[6]Replication package available at https://doi.org/10.5281/zenodo.14057008.

Table 2. Complexity of the Selected Study Subjects

CPS benchmark	Description	Subject	Requirements	#Features	#CFs
ADAS [15]	Advanced driver assistance system with automated lane-keeping and emergency breaking.	ADAS1	3	6	1
		ADAS2	3	6	2
RR [18]	Search and RR (our running example).	RR1	4	9	4
		RR2	4	9	8
UAV [16]	Autonomous team of UAVs carrying out a surveying mission.	UAV1	12	7	3
		UAV2	12	7	6

to crossing pedestrians. The semantic space factors in the scene include vehicle speed, quality of the vision sensor, position and speed of the pedestrians, and road shape, respectively. Both ADAS1 and ADAS2 have 3 requirements and 6 features in total. ADAS2 has two CFs (vehicle speed, vision sensor quality) rather than a single one as ADAS1 (vehicle speed only).

The RR benchmark [18] is the autonomous rover illustrated in Section 2. The rover navigates within a designated search area, aiming to identify injured individuals while making necessary adaptations to meet dependability requirements. In this case, we have four requirements and a bigger semantic space with nine variables in total (see Table 1). Similar to the ADAS benchmark, the second version, RR2, has twice as many CFs as RR1 (eight features compared to four).

UAV [16] represents an autonomous team of UAVs carrying out a surveying mission in a hostile environment. The objective of the team in this scenario is to reach mission targets on the ground (through downward-looking sensors) and, at the same time, avoid threats (via forward-looking sensors). The distance from the team to existing threats affects the likelihood of detecting them, whereas the distance from the team to the ground affects the likelihood of detecting the targets, but also the likelihood of being damaged by a threat. This benchmark has the highest number of requirements (12 requirements), while the semantic space has 7 features in total, including CFs (e.g., flying speed, team formation) and OFs (e.g., weather conditions, number of threats). Again, UAV2 has twice as many CFs as the first version UAV1 (six features compared to three).

The simulation platform adopted in our experiments allows us to control all variables in the semantic space and observe violated requirements in the simulated scenario under the given setting. Further details about the simulator platform as well as the simulated scenarios can be found in the supporting material.

5.1.2 Baseline Methods. For comparison with XDA-II, we use alternative adaptation planning approaches: RS, white-box optimization using XDA, and black-box optimization using NSGA-III and FITEST.

XDA is the predecessor of XDA-II. According to Negri et al. [11], XDA follows a white-box optimization approach by being explanation-driven. However, it relies exclusively on PDPs and, for each adaptation request, performs an additional ranking step. In this step, the algorithm adopts an on-the-fly local feature ranking scheme based on the slopes of the PDP curves. Specifically, the steeper the slope, the higher the feature impact (or importance). Notice that our baseline makes use of NSGA-III to solve the aforementioned optimization problem.

We also adopt traditional black-box optimization methods that recast the problem of adaptation as a constrained many-objective optimization problem. The optimization drives the selection of new system configurations (i.e., assignments to CFs) by increasing the probability of satisfying system-level requirements over a given threshold. Each objective pushes the values of a CF toward the corresponding ideal value. The fitness for each CF is defined as the normalized distance between the actual point (given by the selected adaptation option) and the ideal one. In addition to objectives

we also set constraints in the form $\mathcal{P}_{M_i}(y = true|x) > P$. These constraints define the minimum probability threshold P required for the fulfillment of specific requirements R_i. Thus, each constraint represents the probability estimate expectation set by the corresponding (black-box) classifier M_i. We use this baseline method as it represents a common approach to steer adaptation decisions as described by Kinneer et al. [26] and Camilli et al. [17]. As anticipated above, we solve this many-objective optimization problem using two mainstream algorithms: NSGA-III and FITEST.

In our evaluation, XDA-II comes in two alternative versions leveraging two alternative mainstream techniques to carry out feature ranking. Specifically, we adopt XDA-II-SHAP and XDA-II-PFI which rely on SHAP FI and PFI, respectively.

Finally, we also adopt RS as a baseline method as it provides us with insights into the complexity of the adaptation problem and allows us to quantify the relative cost-effectiveness of XDA-II and the other approaches.

5.1.3 Statistical Tests. To reduce the risk of obtaining results by chance, we account for randomness by executing the adaptation process starting from 200 initial operating conditions using XDA-II and all the selected baseline methods. According to the guideline introduced by Arcuri et al. [27], we apply Mann-Whitney U tests [28] to evaluate the statistical significance of the results. We also measure Vargha and Delaney's $\hat{A}_{AB}$ [29] to compute the effect size of the difference between samples A and B. We adopt the following standard classification: effect size $\hat{A}_{AB}$ ($= 1 - \hat{A}_{BA}$) is small, medium, and large when its value is greater than 0.55, 0.63, and 0.70, respectively.

5.1.4 Evaluation Testbed. All experiments have been executed on a commodity hardware laptop running MacOS Sonoma version 14.5 equipped with an ARM-based Apple M1 Pro processor at 3.2 GHz (8 cores) and 16 GB RAM.

5.2 Results

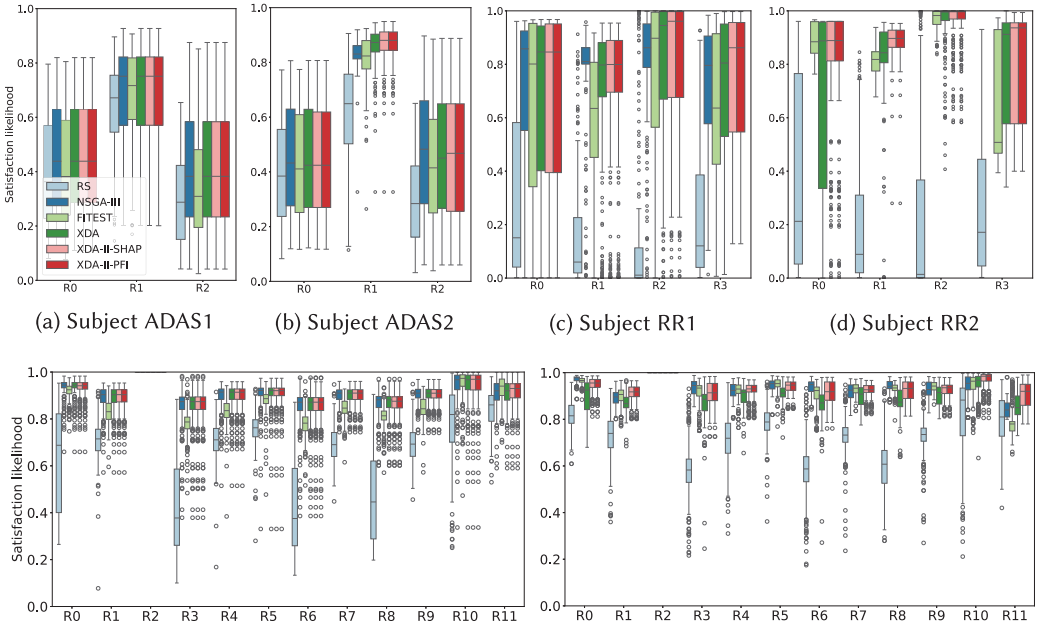
5.2.1 RQ1: Effectiveness of the Decisions.

Setup. To answer RQ1, we execute XDA-II and all selected baseline methods for all evaluation subjects listed in Table 2 using the same set of initial configurations in terms of assignments to variables in the semantic space (200 assignments in total). Each initial assignment represents an operating condition requiring adaptation as it leads to violations of one or more existing requirements. We compare the effectiveness of alternative adaptation planning approaches using two metrics defined as follows.

Metric 1 (Satisfaction Likelihood). Given a requirement R_i, the current value assignment to observable features $\hat{OF}$, and the adaptation decision $\hat{CF^*}$ defined as a new value assignment to CFs, the *satisfaction likelihood* is $\mathcal{P}_{M_i}(y = true|(\hat{CF^*}, \hat{OF}))$, that is, the probability estimate for the class label *true* returned by the black-box classifier M_i with input $(\hat{CF^*}, \hat{OF})$.

Metric 2 (Success Rate). Given a requirement R_i, and a set of operating conditions X, the *success rate* is defined as the number of elements $(\hat{CF^*}, \hat{OF}) \in X$ such that the simulation of the subject under the condition $(\hat{CF^*}, \hat{OF})$ fulfills R_i divided by the total number of elements in X.

For each requirement, the satisfaction likelihood represents the probability anticipated by the corresponding black-box classifier given the operating condition. Instead, the success rate is determined through empirical observations from actual simulations rather than relying on model predictions. This involves observing the subjects across many runs and quantifying the rate of successful adaptation decisions based on the actual satisfaction of requirements. Note that Metric 1 is designed to evaluate the accuracy of adaptation decisions based on the predictive model within the



(a) Subject ADAS1 (b) Subject ADAS2 (c) Subject RR1 (d) Subject RR2

(e) Subject UAV1 (f) Subject UAV2

Fig. 6. Satisfaction likelihood (higher is better).

knowledge component, which, in this specific case, is used as the ground truth. Since explainability does not directly enhance predictive accuracy, the goal of this metric is to assess whether there is no regression in the effectiveness when explainability is incorporated. Metric 2 evaluates the adaptation decisions based on the system's actual behavior, which serves as the ground truth in this case. This allows us to assess how the accuracy of the predictive model (whether explainable or not) translates into actual effectiveness.

To use XDA-II, we need to define the size k of *neighbors* and best candidate solutions. In general, a small number for the parameter k reduces the cost but can also reduce the effectiveness since it limits the exploration of alternative adaptation options. We did not fine-tune this parameter; however, based on our preliminary evaluations, we set $k = 10$, as this value hits a balance by limiting cost overhead while reaching a plateau in effectiveness. To use the baseline methods NSGA-III and FITEST, we tune the parameters (mutation and crossover rates) and the termination condition following the guidelines introduced by the original studies [20, 25]. For NSGA-III, we set the population size equal to the number of reference directions, maintaining the same configuration for FITEST. We also apply the same termination condition, based on fitness convergence tolerance of 10^{-6}, combined with a maximum limit of 100 generations. For all methods under comparison (XDA-II and all baselines), we adopted the same probability threshold $P = 0.8$ for all R_i.

Results. Figure 6 shows the distribution of the satisfaction likelihood for all methods under comparison and all evaluation subjects and requirements. Table 3 shows the results of the statistical tests. Columns A and B indicate the two approaches being compared while columns p-value and $\hat{A}_{AB}$ indicate the statistical significance and effect size, respectively when comparing A and B in terms of estimated satisfaction likelihood. Statistical significance (p-value < 0.05) is highlighted using gray cells, while boldface denotes a large effect size ($\hat{A}_{AB} > 0.7$). Notice that for subject RR2 (Figure 7(d)),

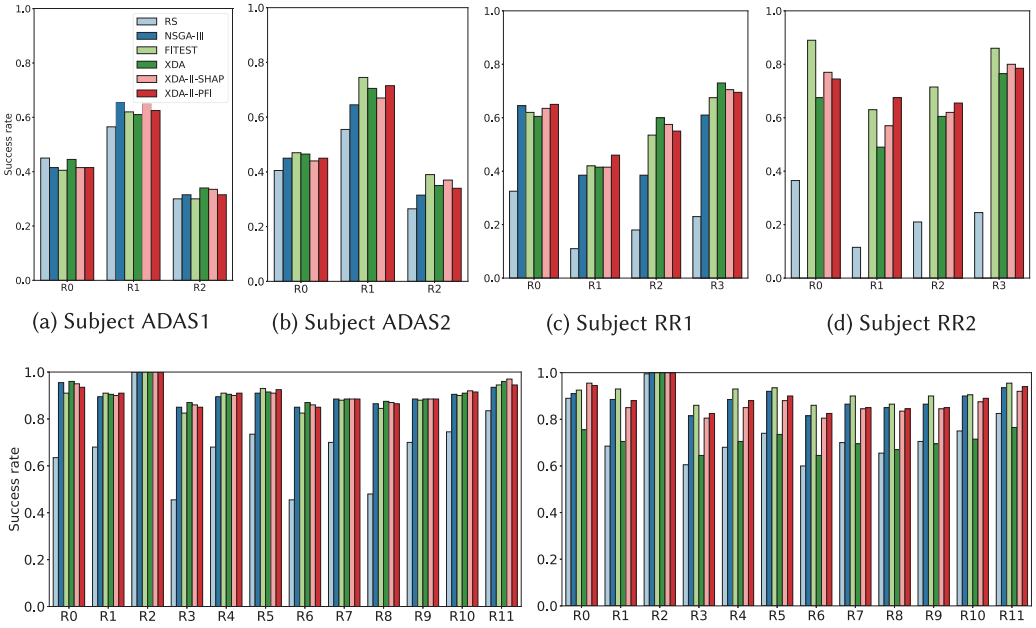
Table 3. Statistical Tests for Satisfaction Likelihood

Subject	Samples		R_0		R_1		R_2		R_3		R_4		R_5		R_6		R_7		R_8		R_9		R_{10}		R_{11}	
	A	B	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$
ADAS1	XDA-II-SHAP	RS	4.74e-27	0.57	3.58e-23	0.63	5.30e-30	0.65	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	NSGA-III	0.28	0.50	0.17	0.49	0.59	0.49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	FITEST	3.75e-23	0.55	0.13	0.51	3.46e-21	0.58	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	XDA	0.28	0.50	0.17	0.49	0.59	0.49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	RS	4.74e-27	0.57	3.58e-23	0.63	5.30e-30	0.65	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	NSGA-III	0.28	0.50	0.17	0.49	0.59	0.49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	FITEST	3.75e-23	0.55	0.13	0.51	3.46e-21	0.58	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	XDA	0.28	0.50	0.18	0.50	0.59	0.49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	XDA-II-PFI	0.50	0.50	0.50	0.50	0.50	0.50	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ADAS2	XDA-II-SHAP	RS	7.13e-20	0.55	1.52e-34	**0.95**	5.37e-34	0.70	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	NSGA-III	3.46e-22	0.48	4.02e-14	**0.76**	7.80e-20	0.47	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	FITEST	7.15e-4	0.51	2.80e-19	**0.73**	3.06e-12	0.54	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	XDA	1.52e-3	0.49	1.89e-05	0.54	0.37	0.49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	RS	7.13e-20	0.60	1.52e-34	**0.96**	5.37e-34	0.70	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	NSGA-III	3.46e-22	0.48	4.02e-14	**0.76**	7.80e-20	0.47	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	FITEST	7.15e-4	0.52	2.80e-19	**0.73**	3.07e-12	0.54	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	XDA	1.52e-3	0.49	1.89e-05	0.54	0.34	0.49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	XDA-II-PFI	0.50	0.50	0.49	0.49	0.49	0.49	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RR1	XDA-II-SHAP	RS	1.71e-27	**0.72**	1.76e-31	**0.85**	5.06e-34	**0.91**	4.49e-34	**0.90**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	NSGA-III	1.62e-3	0.49	3.30e-07	0.43	**0.88**	0.61	0.41	0.56	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	FITEST	2.16e-11	0.53	1.86e-21	0.66	3.91e-06	0.56	1.12e-25	0.67	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	XDA	0.51	0.50	0.49	0.50	0.13	0.51	9.25e-6	0.54	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	RS	1.71e-27	**0.73**	1.76e-31	**0.86**	5.06e-34	**0.90**	4.49e-34	**0.90**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	NSGA-III	1.62e-3	0.49	3.31e-07	0.43	**0.88**	0.62	0.41	0.56	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	FITEST	2.13e-11	0.51	1.86e-21	0.67	3.92e-06	0.57	1.12e-25	0.67	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	XDA	0.52	0.50	1.76e-31	0.51	0.13	0.51	9.25e-06	0.54	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	XDA-II-PFI	0.49	0.50	0.50	0.50	0.50	0.50	0.51	0.50	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RR2	XDA-II-SHAP	RS	7.34e-24	**0.83**	1.43e-34	**0.99**	1.14e-33	**0.93**	3.28e-34	**0.92**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	NSGA-III	NA	NA	NA	NA	NA	NA	NA	NA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	FITEST	7.95e-2	0.51	1.82e-30	**0.86**	0.15	0.59	2.51e-29	**0.76**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	XDA	5.69e-5	0.58	1.4e-15	0.66	0.41	0.51	1.19e-3	0.54	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	RS	7.33e-24	**0.82**	1.43e-34	**0.99**	1.15e-33	**0.93**	3.28e-34	**0.92**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	NSGA-III	NA	NA	NA	NA	NA	NA	NA	NA	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	FITEST	7.95e-2	0.51	1.82e-30	**0.86**	0.15	0.59	2.50e-29	**0.76**	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-PFI	XDA	5.69e-5	0.58	1.42e-15	0.66	0.42	0.51	1.19e-3	0.54	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	XDA-II-SHAP	XDA-II-PFI	0.50	0.50	0.49	0.49	0.49	0.49	0.50	0.51	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UAV1	XDA-II-SHAP	RS	1.08e-33	**0.96**	9.03e-34	**0.95**	0.50	0.50	4.25e-34	**0.97**	6.99e-34	**0.95**	1.18e-33	**0.95**	4.25e-34	**0.97**	2.09e-34	**0.98**	2.83e-34	**0.98**	2.83e-34	**0.97**	1.96e-26	**0.86**	9.41e-20	**0.77**
	XDA-II-SHAP	NSGA-III	5.95e-05	0.45	0.30	0.50	0.50	0.50	0.28	0.49	0.18	0.49	8.41e-3	0.49	0.28	0.50	0.95	0.49	5.94e-05	0.47	1.47e-3	0.49	8.53e-2	0.50	1.84e-4	0.51
	XDA-II-SHAP	FITEST	8.39e-13	0.69	2.59e-26	**0.81**	0.50	0.50	2.34e-22	**0.83**	4.42e-25	**0.82**	1.12e-24	**0.77**	6.10e-23	**0.83**	3.86e-27	**0.82**	1.09e-24	**0.81**	4.05e-28	**0.82**	7.15e-2	0.47	4.92e-06	0.39
	XDA-II-SHAP	XDA	5.95e-05	0.46	0.31	0.50	0.50	0.50	0.28	0.49	0.18	0.49	8.41e-3	0.49	0.28	0.50	**0.95**	0.49	5.94e-05	0.47	1.47e-3	0.49	8.53e-2	0.50	1.84e-4	0.51
	XDA-II-PFI	RS	1.08e-33	**0.96**	9.03e-34	**0.97**	0.50	0.50	4.25e-34	**0.97**	6.99e-34	**0.95**	1.18e-33	**0.95**	4.25e-34	**0.97**	2.09e-34	**0.98**	2.83e-34	**0.98**	2.83e-34	**0.97**	1.96e-26	**0.86**	9.41e-20	**0.77**
	XDA-II-PFI	NSGA-III	5.95e-05	0.46	0.30	0.51	0.50	0.50	0.28	0.49	0.18	0.49	8.41e-3	0.49	0.28	0.50	**0.95**	0.49	5.94e-05	0.47	1.47e-3	0.49	8.53e-2	0.50	1.84e-4	0.52
	XDA-II-PFI	FITEST	8.39e-13	0.69	2.59e-26	**0.81**	0.50	0.50	2.34e-22	**0.83**	4.42e-25	0.82	1.11e-24	**0.77**	6.10e-23	**0.83**	3.86e-27	**0.82**	1.09e-24	**0.81**	4.05e-28	**0.82**	7.15e-2	0.47	4.92e-06	0.39
	XDA-II-PFI	XDA	5.95e-05	0.45	0.31	0.50	0.50	0.50	0.28	0.49	0.18	0.49	8.41e-3	0.49	0.28	0.50	**0.95**	0.49	5.94e-05	0.47	1.47e-3	0.49	8.53e-2	0.50	1.84e-4	0.51
	XDA-SHAP	XDA-PFI	0.50	0.50	0.49	0.50	0.50	0.50	0.50	0.50	0.49	0.49	0.50	0.50	0.50	0.50	0.49	0.49	0.50	0.51	0.49	0.51	0.50	0.50	0.50	0.50
UAV2	XDA-II-SHAP	RS	2.95e-34	**0.96**	1.96e-34	**0.97**	0.50	0.50	1.74e-34	**0.98**	1.99e-34	**0.98**	1.94e-34	**0.97**	1.96e-34	**0.98**	1.61e-34	**0.98**	1.72e-34	**0.98**	1.82e-34	**0.98**	6.44e-30	**0.90**	1.89e-26	**0.83**
	XDA-II-SHAP	NSGA-III	1.20e-19	0.22	6.59e-8	0.65	0.50	0.50	1.51e-8	0.35	0.32	0.53	2.11e-2	0.43	4.37e-6	0.39	4.79e-2	0.52	2.92e-8	0.37	0.90	0.47	2.42e-12	0.71	5.97e-23	**0.79**
	XDA-II-SHAP	FITEST	4.80e-13	0.31	4.77e-4	0.60	0.50	0.50	8.78e-2	0.48	0.98	0.50	3.07e-4	0.38	0.27	0.56	0.11	0.46	5.93e-3	0.48	4.42e-6	0.36	1.19e-8	0.70	1.19e-26	**0.89**
	XDA-II-SHAP	XDA	9.94e-15	0.68	1.33e-25	**0.80**	0.50	0.50	3.00e-11	0.67	3.48e-15	**0.72**	1.07e-18	**0.76**	2.70e-17	**0.72**	2.34e-12	**0.72**	1.19e-17	**0.72**	1.69e-11	0.69	3.42e-8	0.65	3.72e-15	**0.73**
	XDA-II-PFI	RS	2.95e-34	**0.98**	1.96e-34	**0.97**	0.50	0.50	1.74e-34	**0.99**	1.99e-34	**0.96**	1.94e-34	**0.97**	1.97e-34	**0.98**	1.61e-34	**0.98**	1.72e-34	**0.98**	1.80e-34	**0.99**	6.45e-30	**0.89**	1.89e-26	**0.82**
	XDA-II-PFI	NSGA-III	1.20e-19	0.22	6.59e-8	0.65	0.50	0.50	1.51e-8	0.36	0.38	0.52	2.11e-2	0.43	4.37e-6	0.40	4.79e-2	0.52	2.93e-8	0.36	0.90	0.47	2.42e-12	**0.72**	5.99e-23	0.81
	XDA-II-PFI	FITEST	4.82e-13	0.31	4.77e-4	0.61	0.50	0.50	8.78e-2	0.48	0.98	0.51	3.07e-4	0.38	0.27	0.56	0.12	0.46	5.93e-3	0.50	4.40e-6	0.36	1.17e-8	0.70	1.18e-26	0.89
	XDA-II-PFI	XDA	9.94e-15	0.67	1.35e-25	**0.80**	0.50	0.50	3.02e-11	0.67	3.49e-15	**0.73**	1.07e-18	**0.76**	2.71e-17	**0.72**	2.35e-12	**0.72**	1.19e-17	**0.73**	1.71e-11	0.66	3.40e-8	0.62	3.72e-15	**0.75**
	XDA-II-SHAP	XDA-II-PFI	0.50	0.50	0.50	0.50	0.50	0.50	0.49	0.50	0.51	0.50	0.50	0.50	0.48	0.50	0.50	0.50	0.49	0.49	0.50	0.50	0.50	0.50	0.51	0.50

Statistical significance (p-value < 0.05) is highlighted using gray cells, while boldface denotes a large effect size ($\hat{A}_{AB} > 0.7$).

NSGA-III results are missing for all requirements, leading to Not Available (NA) entries in the corresponding statistical tests in Table 3. This is because, in this case, NSGA-III cannot compute solutions within the given budget (memory usage limit). We provide a detailed discussion of these aspects in the context of RQ3 (cost of the online stage).

Overall, we observe that in cases where the optimization problem is relatively straightforward (such as with all requirements for ADAS1 and R_0 for ADAS2), the differences between the methods become negligible. The distributions are almost identical, and statistical tests do not reveal any significant effect size. When the optimization problem becomes harder (as with subjects RR1, RR2, UAV1, and UAV2), evidenced by the significantly lower performance of the baseline RS, XDA-II (using both SHAP and PFI) can reach relatively higher median values than the other baseline methods. Compared to NSGA-III, FITEST, and XDA, XDA-II yields a higher median in 40%, 66%, and 59% of the cases, respectively. However, as shown in Table 3, no single method demonstrates clear statistical dominance across all subjects and requirements. Considering again non-trivial subjects (RR1, RR2, UAV1, and UAV2), XDA-II is significantly better (with a large effect size) than NSGA-III, FITEST, and XDA in 14%, 60%, and 50% of the cases, respectively. In all other cases, the differences observed are negligible to small. Moreover, when comparing XDA-II-SHAP and XDA-II-PFI, no significant differences are evident. Based on our results regarding satisfaction likelihood, we can conclude that XDA-II is at least as effective as the selected baseline methods, including both white-box (XDA) and black-box approaches (NSGA-III and FITEST).



(a) Subject ADAS1 (b) Subject ADAS2 (c) Subject RR1 (d) Subject RR2

(e) Subject UAV1 (f) Subject UAV2

Fig. 7. Success rate (higher is better).

We further investigate the effectiveness of XDA-II by studying the success rate for all requirements and all evaluation subjects reported in Figure 7. According to Metric 2, we extract this data by simulating all subjects after actuating the adaptation decisions calculated by all the methods under comparison starting from the same set of initial operating conditions.

We observe that for both relatively simpler subjects (ADAS1 and ADAS2) and more complex ones (RR1, RR2, UAV1, and UAV2), XDA-II-SHAP and XDA-II-PFI achieve comparable success rates. These rates are also very close to those attained by NSGA-III and FITEST across all requirements. For the more complex subjects (RR1, RR2, UAV1, and UAV2), when the complexity of the optimization problem increases by doubling the number of controllable factors, the effectiveness of the predecessor, XDA, declines compared to XDA-II. Specifically, for the RR benchmark, we observe an average effectiveness increase of 3.9% between RR1 and RR2. Similarly, for the UAV benchmark, we see an average increase of 9.8% between UAV1 and UAV2. Overall, XDA-II proves to be as effective as other black-box optimization methods (NSGA-III and FITEST) and more effective than XDA, particularly in terms of the actual success of adaptation decisions.

RQ1 Summary. In terms of satisfaction likelihood, XDA-II (both SHAP and PFI versions) is at least as effective as the selected baseline methods, including both white-box (XDA) and black-box approaches (NSGA-III and FITEST). Regarding success rates, XDA-II-SHAP and XDA-II-PFI yield comparable outcomes, with values closely matching those of NSGA-III and FITEST. XDA-II outperforms XDA, particularly as the complexity of the subject increases, where we observe an average improvement of up to 9.8%.

Table 4. Statistical Tests for Quality Score

Comparison		ADAS1		ADAS2		RR1		RR2		UAV1		UAV2	
A	*B*	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$
XDA-II-SHAP	NSGA-III	0.50	0.50	2.14e-6	0.43	4.53e-6	0.65	NA	NA	1.74e-2	0.53	2.06e-2	0.56
XDA-II-SHAP	FITEST	4.00e-22	**0.75**	2.97e-4	0.46	2.21e-20	**0.77**	1.11e-20	**0.82**	9.01e-9	0.34	9.48e-24	0.16
XDA-II-SHAP	XDA	0.50	0.50	6.97e-2	0.51	3.23e-7	0.58	1.22e-18	**0.71**	1.74e-2	0.52	2.73e-26	**0.84**
XDA-II-PFI	NSGA-III	0.50	0.50	2.16e-6	0.44	4.51e-6	0.66	NA	NA	1.75e-2	0.53	2.06e-2	0.55
XDA-II-PFI	FITEST	4.05e-22	**0.75**	2.95e-4	0.45	2.05e-19	**0.76**	1.09e-20	**0.82**	9.12e-9	0.35	9.12e-23	0.18
XDA-II-PFI	XDA	0.50	0.50	6.99e-2	0.51	3.20e-7	0.56	1.17-18	**0.72**	1.71e-2	0.50	2.75e-26	**0.85**
XDA-SHAP	XDA-PFI	0.50	0.50	0.50	0.49	0.50	0.49	0.50	0.50	0.49	0.50	0.50	0.50

Statistical significance (p-value < 0.05) is highlighted using gray cells, while boldface denotes a large effect size ($\hat{A}_{AB} > 0.7$).

5.2.2 *RQ2: Quality of the Adaptations.*

Setup. To answer RQ2, we use the same setup as in RQ1 but we measure the quality of the adaptation decisions calculated by XDA-II and our selected baseline methods. We compute and then compare the quality starting from all initial conditions given in terms of assignments to variables in the semantic space (200 assignments in total). The quality of a given adaptation decision is defined as follows.

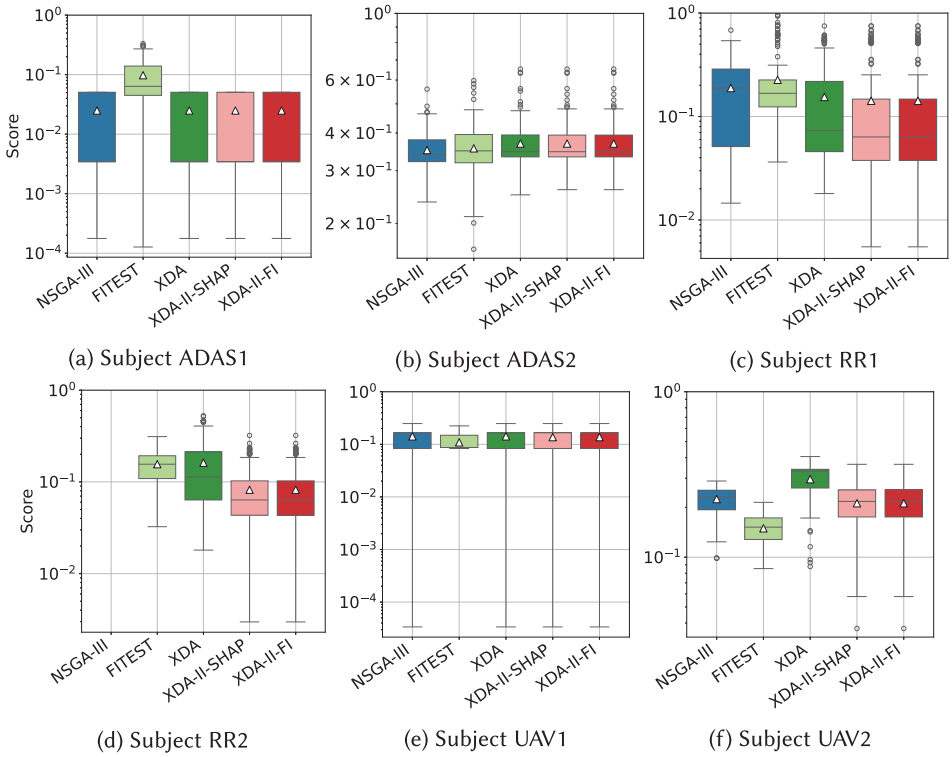
Metric 3 (Quality Score). Given the adaptation decision $\hat{CF}^*$ defined as a new value assignment to CFs, the *quality score* is $d_{\text{norm}}(\hat{CF}^*, I)$, that is, the normalized Euclidean distance between the chosen configuration and the corresponding ideal operating condition I.

The quality score d_{norm} quantifies how closely the post-adaptation operating conditions align with the ideal conditions (the smaller the distance, the better). We use normalization to adjust scale differences in the CFs. Namely, we ensure that each feature contributes equally to the distance calculation, regardless of the original scale.

Results. Figure 8 shows the distribution of the quality score for all methods under comparison and all evaluation subjects. Table 4 shows the results of the corresponding statistical tests.[7] Columns *A* and *B* indicate the two approaches being compared, while columns p-value and $\hat{A}_{AB}$ indicate the statistical significance and effect size computed for all subjects. As anticipated in RQ1, NSGA-III results are missing for subject RR2 (Figure 8(d)), leading to NA entries in the corresponding statistical tests in Table 4.

We observe that XDA-II (both SHAP and PFI versions) generally achieve similar or superior quality scores compared to other baseline methods, with one exception: in the case of subject UAV2, where FITEST outperforms XDA-II, which is still significantly better than NSGA-III (medium effect size) and XDA (large effect size). XDA-II-SHAP and XDA-II-PFI show similar performance. As shown in Table 4, there is no statistical significance, and the effect size is negligible across all subjects in this case. XDA-II outperforms other baseline methods as the complexity of the evaluation subject increases, particularly with larger semantic spaces and a higher number of CFs. For RR, the benchmark with the highest number of features, the scores of both XDA-II-SHAP and XDA-II-PFI are significantly better than those of all other methods. In RR1 (with 4 CFs), comparisons with FITEST show a large effect size, while comparisons with other methods show a medium effect size.

[7] Notice that we exclude RS from the baseline comparisons as, in this case, it would be an uninformative benchmark. Random operating conditions may, by chance, yield better scores (closer proximity to ideal values); however, these scores are unreliable and often correspond to incorrect solutions that fail to meet one or more requirements.



(a) Subject ADAS1 (b) Subject ADAS2 (c) Subject RR1

(d) Subject RR2 (e) Subject UAV1 (f) Subject UAV2

Fig. 8. Quality score (lower is better).

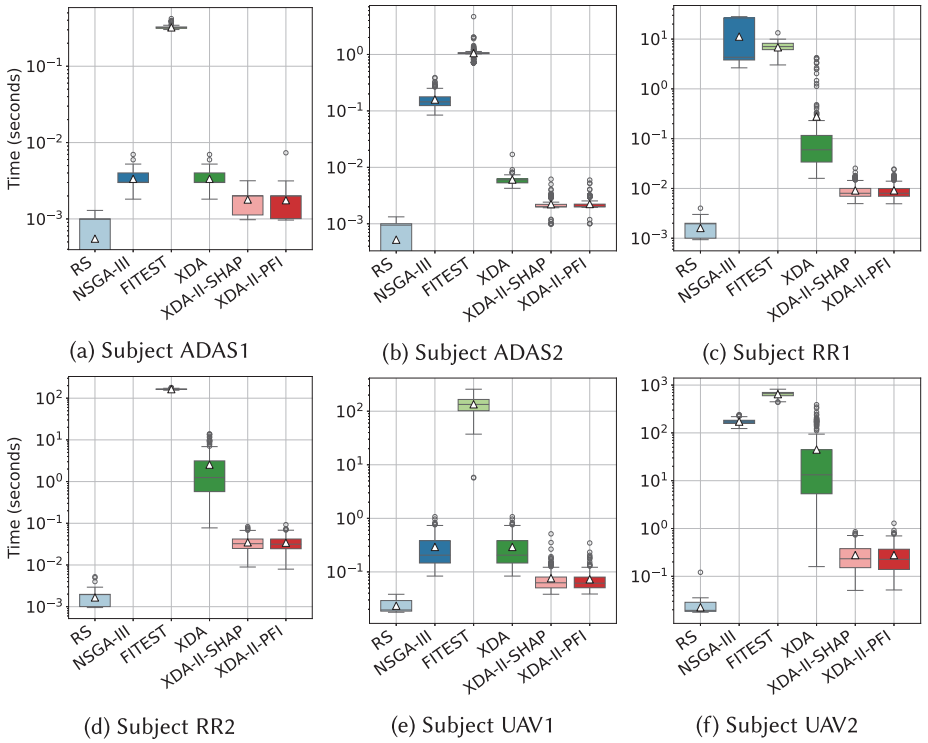
In RR2 (with 8 CFs), the effect size is large in all comparisons except with NSGA-II, as data for this case is unavailable.

RQ2 Summary. XDA-II, in both SHAP and PFI versions, generally achieves higher quality scores than other baseline methods, with significant improvements as the complexity of the evaluation subject increases, particularly in larger semantic spaces and with more CFs. For RR, the benchmark with the highest number of features, statistical tests reveal that XDA-II consistently outperforms all baselines with medium to large effect size.

5.2.3 *RQ3: Cost of the Online Stage.*

Setup. To answer RQ3, we use the same setup as in RQ1 and RQ2 but we measure the cost as the wall-clock execution time and memory consumption required by the plan component. We compute the cost starting from each initial condition given in terms of assignments to variables in the semantic space (200 assignments in total) using the same budget defined as execution time upper-bound equal to 1,000 seconds. Then, we compare the results obtained using XDA-II and all the selected baseline methods.

Results. Figure 9 shows the distribution of the execution time for all methods across each evaluation subject. We observe that XDA-II (in both SHAP and PFI versions) consistently yields lower costs than all other baseline methods, except for RS, which assigns random values to CFs (i.e., random adaptation decisions). Table 5 shows the results of the corresponding statistical tests.



(a) Subject ADAS1 (b) Subject ADAS2 (c) Subject RR1

(d) Subject RR2 (e) Subject UAV1 (f) Subject UAV2

Fig. 9. Cost in terms of wall-clock execution time (lower is better).

Table 5. Statistical Tests for Execution Time (Online Stage)

Comparison		ADAS1		ADAS2		RR1		RR2		UAV1		UAV2	
A	B	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$
XDA-II-SHAP	RS	1.37e-33	0.06	1.52e-34	0.01	1.43e-34	0.0	1.43e-34	0.0	1.43e-34	0.0	1.43e-34	0.0
XDA-II-SHAP	NSGA-III	1.90e-33	**0.91**	1.43e-34	**1.0**	1.43e-34	**1.0**	NA	NA	1.18e-33	**0.96**	1.43e-34	**1.0**
XDA-II-SHAP	FITEST	1.45e-34	**0.99**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**
XDA-II-SHAP	XDA	1.90e-33	**0.91**	1.43e-34	**0.99**	1.43e-34	**0.99**	1.43e-34	**0.99**	1.18e-33	**0.96**	1.52e-34	**0.97**
XDA-II-PFI	RS	6.91e-32	0.09	1.43e-34	0.02	1.43e-34	0.0	1.43e-34	0.0	1.43e-34	0.0	1.43e-34	0.0
XDA-II-PFI	NSGA-III	9.46e-33	**0.91**	1.43e-34	**1.0**	1.43e-34	**1.0**	NA	NA	1.43e-34	**0.97**	1.43e-34	**1.0**
XDA-II-PFI	FITEST	1.45e-34	**0.99**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**
XDA-II-PFI	XDA	9.46e-33	**0.91**	1.50e-34	**0.99**	1.43e-34	**0.99**	1.43e-34	**0.99**	1.43e-34	**0.97**	1.54e-34	**0.97**
XDA-SHAP	XDA-PFI	0.51	0.47	0.42	0.53	0.42	0.50	1.51e-06	0.48	0.21	0.50	0.01	0.48

Statistical significance (p-value < 0.05) is highlighted using gray cells, while boldface denotes a large effect size ($\hat{A}_{AB} > 0.7$).

We can also observe that when increasing the complexity of the study subject (by doubling the number of CFs), both XDA-II-SHAP and XDA-II-PFI show a smaller cost increase compared to NSGA-III, FITEST, and XDA. Specifically, in the RR benchmark (with the highest number of features), the median cost for XDA and FITEST rises by over an order of magnitude when moving from RR1 (4 CFs) to RR2 (8 CFs), while NSGA-III fails to complete within the allocated budget (1,000 seconds). In contrast, XDA-II shows a median cost increase of about 0.5 order of magnitude from RR1 to RR2. The median execution time of XDA-II stays below 1 second across all subjects. For the other methods (excluding RS), the median execution time increases to up to 10 seconds for XDA and exceeds 100 seconds for both FITEST and NSGA-III.

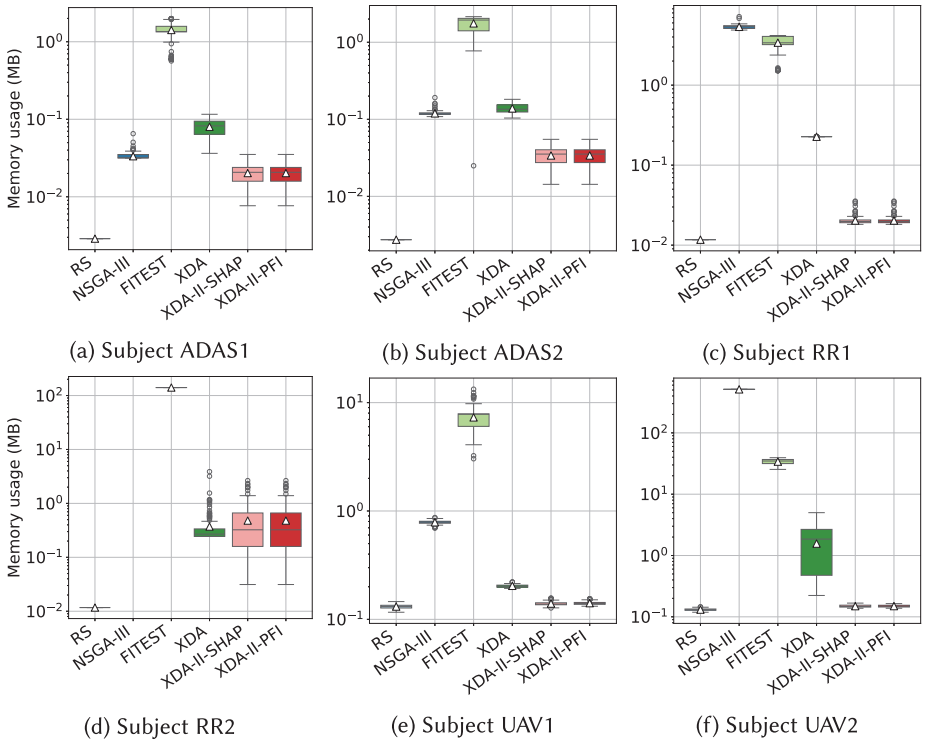


Fig. 10. Cost in terms of memory consumption (lower is better).

As shown in Table 5, the comparison between XDA-II-SHAP and XDA-II-PFI reveals negligible differences for all study subjects. Comparisons between either XDA-II-SHAP or XDA-II-PFI and all other baselines, except RS, yield statistically significant results with large effect sizes.

Figure 10 reports the cost in terms of memory consumption, while Table 6 reports corresponding statistical tests. Similar trends are observed for memory consumption. Specifically, XDA-II-SHAP and XDA-II-PFI show comparable memory usage, as confirmed by statistical tests. Both versions of XDA-II consistently outperform the predecessor, XDA, except in RR2, where their costs are similar. Compared to the other baseline methods (excluding RS), XDA-II demonstrates significantly lower memory costs with a large effect size across all study subjects.

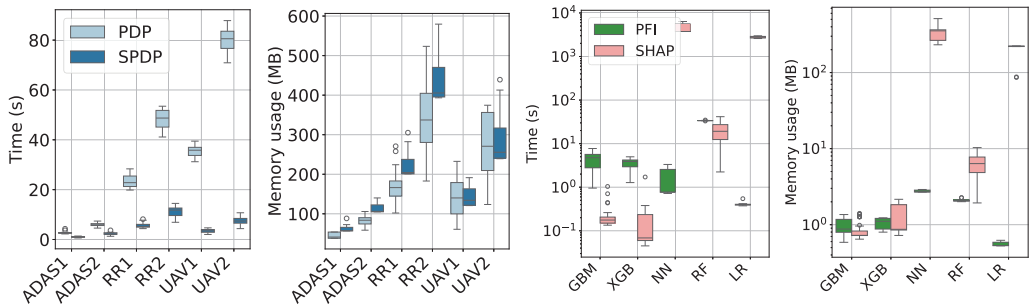
Results show that XDA-II has significant advantages compared to XDA and other mainstream black-box optimization methods, specifically in terms of cost (execution time and memory consumption). This aspect is often crucial, especially within self-adaptive systems running in critical domains, exemplified by our study subjects.

RQ3 Summary. XDA-II (in both SHAP and PFI versions) consistently demonstrates lower execution time and memory consumption than all other baseline methods except RS, with statistically significant differences and large effect sizes across all study subjects. As the complexity of the study subject increases, XDA-II maintains a lower cost increase compared to XDA, FITEST, and NSGA-III, demonstrating its suitability for resource-sensitive, self-adaptive systems in critical domains.

Table 6. Statistical Tests for Memory Consumption (Online Stage)

Comparison		ADAS1		ADAS2		RR1		RR2		UAV1		UAV2	
A	*B*	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$	p-val	$\hat{A}_{AB}$
XDA-II-SHAP	RS	1.40e-34	0.0	1.42e-34	0.0	1.43e-34	0.0	1.43e-34	0.0	6.42e-29	0.11	1.59e-34	0.0
XDA-II-SHAP	NSGA-III	1.93e-34	**0.98**	1.43e-34	**1.0**	1.43e-34	**1.0**	NA	NA	1.18e-33	**1.0**	1.43e-34	**1.0**
XDA-II-SHAP	FITEST	1.43e-34	**1.0**	1.43e-34	**0.99**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**
XDA-II-SHAP	XDA	1.38e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**0.99**	1.43e-34	**0.99**	1.18e-33	**1.0**	1.43e-34	**1.0**
XDA-II-PFI	RS	1.40e-34	0.0	1.42e-34	0.0	1.43e-34	0.0	1.43e-34	0.0	3.18e-31	0.07	1.57e-34	0.0
XDA-II-PFI	NSGA-III	1.93e-34	**0.98**	1.43e-34	**1.0**	1.43e-34	**1.0**	NA	NA	1.43e-34	**1.0**	1.43e-34	**1.0**
XDA-II-PFI	FITEST	1.43e-34	**1.0**	1.43e-34	**0.99**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**1.0**
XDA-II-PFI	XDA	1.38e-34	**1.0**	1.43e-34	**1.0**	1.43e-34	**0.99**	1.43e-34	**0.99**	1.43e-34	**1.0**	1.43e-34	**1.0**
XDA-II-SHAP	XDA-II-PFI	0.31	0.49	0.31	0.49	0.06	0.49	0.03	0.48	2.53e-4	0.58	0.40	0.52

Statistical significance (p-value < 0.05) is highlighted using gray cells, while boldface denotes a large effect size ($\hat{A}_{AB} > 0.7$).



(a) PDP/SPDP time (b) PDP/SPDP memory (c) Feature rank time (d) Feature rank memory

Fig. 11. Cost overhead of the offline stage (lower is better).

5.2.4 *RQ4: Cost Overhead of the Offline Stage.*

Setup. To address RQ4, we execute the offline stage 20 times for each study subject, measuring cost overhead in terms of wall-clock execution time and memory consumption required for calculating PDP/SPDP explanations and generating feature rankings using SHAP and PFI by changing the predictive models adopted by the subject. In particular, we use mainstream classifiers including **Gradient Boosting Machine (GBM)**, **eXtreme Gradient Boosting (XGB)**, NN, RFs, and **Logistic Regression (LR)**. Note that this offline stage is unique to XDA-II and its predecessor, XDA, as the other baseline methods do not require it. Specifically, XDA requires only PDP/SPDP explanations and does not perform feature ranking.

Results. Figure 11 shows the wall-clock execution time and memory consumption required to compute the PDP/SPDP explanations (Figure 11(a) and (b), respectively) as well as the execution time and memory consumption required for feature ranking (Figure 11(c) and (d), respectively).

The execution time and memory required to compute PDPs and SPDPs depend on the subject complexity, specifically the size of the semantic space and the number of requirements. For PDPs, the median execution time ranges from a few milliseconds (for ADAS1 and ADAS2) to approximately 50 and 80 seconds (for UAV2 and RR2, respectively). In contrast, SPDP creation is cheaper, with execution time spanning from a few milliseconds to a maximum of 10 seconds. Memory usage remains relatively low across all subjects, with median consumption ranging from roughly 100 MB (ADAS1 and ADAS2) to 300–400 MB (for RR2 and UAV2, respectively).

Feature ranking complexity is influenced primarily by the explanation technique used and the predictive model type. Overall, SHAP yields higher costs than PFI for NN and LR models, while costs are comparable across other models. For FI, the median execution time ranges from 1 to 10 seconds for GBM, XGB, NN, and LR, increasing to about 30 seconds for RF. SHAP execution times vary from a few milliseconds (GBM, XGB) to approximately 5,000 seconds for NN. Memory consumption follows a similar pattern: for FI, median values range from around 1 MB (GBM, XGB, LR) to about 5 MB (NN, RF), while for SHAP, they range from 1 MB (GBM, XGB) to 10 MB (RF) and up to 500 MB for NN and LR.

Notice that the offline stage runs only once before deploying the system in its target environment, so the cost overhead described above does not impact runtime performance. Instead, as described in RQ3, it provides additional information that helps reduce runtime costs, enabling the system to respond and adapt more efficiently.

RQ4 Summary. The cost of computing PDPs and SPDPs varies with subject complexity, with median execution times ranging from milliseconds (ADAS1, ADAS2) to around 50–80 seconds (UAV2, RR2), while memory usage remains low, between 100 and 400 MB. Feature ranking costs differ by method and model type; SHAP incurs higher costs than PFI for NN and LR models, while both methods require minimal runtime overhead as the offline stage runs once before deployment, providing insights that optimize runtime adaptation.

6 Discussion

We discuss major validity threats, corresponding mitigation actions, and then the advantages and disadvantages of our approach XDA-II.

6.1 Threats to Validity

For practical reasons, we conducted our experiments by simulating the evaluation subjects rather than using real systems. Using one simulation platform is a potential threat to external validity. To mitigate this issue, we consider multiple (6) evaluation subjects having increasing complexity in terms of number of requirements (from 3 to 12), size of the semantic space (from 6 to 9), and number of CFs (from 1 to 8). Subjects are derived from the existing literature and they are open source, thus, they can be reused by other researchers for comparison. For each evaluation subject, we control the effect of changing factors to avoid trivial operating conditions (i.e., requirements cannot be violated or requirements are always violated). Note that such a fine-grained manipulation increases the internal validity of our results compared to observations without manipulation. Further details about the simulation platform and the evaluation subjects are publicly available in our replication package. Additional studies using more than one simulator and study subjects from other domains would increase the generalizability of our results.

To avoid the risk of obtaining results by chance, we repeated our experiments using a large sample of initial configurations (200 assignments in total) for all subjects and methods under comparison. The statistical significance and effect size of our results have been assessed using the Mann-Whitney U test and Vargha and Delaney's $\hat{A}_{AB}$ measure according to the guidelines introduced by Arcuri et al. [27].

6.2 Advantages and Disadvantages

According to our experience, XDA-II can be effectively used to guide adaptation decisions through global explanations. Its adoption offers the following advantages.

— Our approach is many-objective, meaning that the adaptation process takes into account multiple requirements.
— We rely on a combination of model-agnostic interpretable ML techniques. This means that XDA-II is flexible and allows for consistent interpretability across various types of predictive models, as shown in our empirical evaluation.
— In addition to increased efficiency, our approach makes the adaptation process interpretable for humans via explanations. This typically enhances comprehension and significantly simplifies debugging of the control loop.
— We adopt white-box optimization that tends to be more efficient than optimizing black-box functions due to the transparency of the internal structure of the functions and their properties.

We also identified the following potential drawbacks.

— XDA-II requires an (offline) extra step to calculate all the required explanations out of the black-box classifiers contained in the knowledge component (PDPs, SHAP, and PFI in our case). According to our results, this does not yield a significant overhead since the step is executed once before deployment. Nevertheless, this extra cost may increase with the number of requirements and size of the semantic space.
— Climbing and descending steps heavily depend on individual conditional expectation curves built from the nearest neighborhood. In cases where the actual operating conditions within that neighborhood significantly differ, the effectiveness of our approach may decrease.
— Significant shifts in the environment or system behavior (i.e., concept drift) can degrade model accuracy, necessitating retraining. This is a well-known tradeoff between the efficiency of pre-trained models and the adaptability of online learning techniques. Note that this is not a limitation specific to XDA-II, but rather an inherent characteristic of any self-adaptive system that relies on pre-trained predictive models to support decision-making.

7 Related Work

We discuss related work in three main areas. We review foundational research in explainability. Then, we examine how explainability has been affecting software engineering. Finally, we focus on existing approaches at the intersection of explainability and self-adaptive systems, drawing attention to the specific challenges these approaches address and the remaining open issues.

7.1 Foundations of Explainability

Explainability has become central to the development of trustworthy AI systems driven by black-box models such as NNs or ensemble methods. Angelov et al. [30] highlight that opacity in predictive systems can undermine user confidence and hamper effective decision-making. Thus, ensuring transparency is crucial, particularly in safety-critical contexts.

Efforts to formalize and structure XAI challenges span multiple disciplines. A recent manifesto [31] outlines open problems (ranging from methodological gaps to scalability issues) and proposes strategies for addressing them. Concurrently, Langer et al. [32] define a comprehensive model for designing and evaluating explainability techniques that accommodate the needs of diverse stakeholders (e.g., data scientists, domain experts, and users). Nauta et al. [33] introduce a quantitative evaluation method, enabling systematic assessments and comparisons across different explanations.

Advantages of foundational research in explainability include specific explainability methods, conceptual frameworks, and guidelines for evaluation. One key *limitation* of these approaches is that they remain mainly model-centric and do not fully address how explainability shall be integrated into broader software lifecycles, particularly when facing the complexity of large-scale applications or stringent operational constraints.

7.2 Explainability in Software Engineering

Although explainability originated primarily in AI research, the software engineering community has also grown interested in improving the interpretability of AI-based software components and analyzing explainability as a quality attribute. Relevant examples include exploring explainable analytical models for decision-making and predictions [34], explainable counterexamples in verification, such as model checking [35], and explainable trade-offs among quality attributes to guide architectural choices [36].

Chazette et al. [37] study how explainability affects requirements elicitation and validation, identifying explainability concerns as key non-functional requirements that may conflict with other non-functional requirements (e.g., performance, security). Blumreiter et al. [38] propose a general framework for plugging *self-explainable* capabilities into software systems already deployed in production. The goal is to improve trust by allowing engineers (and sometimes end-users) to understand the rationale behind opaque system decisions.

The *advantages* of techniques like explainable counterexamples or dashboards for quality-attribute tradeoffs enhance transparency and enable more informed decision-making across the software lifecycle. Furthermore, treating explainability as a non-functional requirement clarifies where and how these explanatory mechanisms should be engineered and enforced, enforcing alignment with other system-level objectives. Current *limitations* of these approaches include their narrow applicability, often tailored to specific domains or confined to particular stages of the software lifecycle. Furthermore, the trade-off between explainability and other quality attributes (e.g., performance, security) is typically addressed in an ad hoc manner, due to the lack of systematic guidelines for integrating and balancing explainability within broader requirements engineering practices.

7.3 Explainability in Autonomous and Adaptive Systems

Autonomous and adaptive systems continually reconfigure themselves to cope with changes in the environment, resource availability, or user needs. When such adaptations rely on opaque predictive models, humans may struggle to understand why certain adaptations are enacted. Consequently, integrating explainability principles into self-adaptation mechanisms has received growing attention.

Diallo et al. [39] adopt explainability to reduce the adaptation space, thereby making adaptation decisions more efficient and interpretable for human stakeholders. Li et al. [40] highlight that the usefulness of explanations is not always self-evident and depends on the context and the type of explanation provided. They introduce a formal framework for reasoning about when explanations are warranted and how they can meaningfully support human understanding of adaptive system behavior. Wohlrab et al. [41] adopt dimensionality reduction, clustering, and interpretable models (e.g., decision trees) to uncover and communicate the rationale behind automated planning decisions. Similarly, Feit et al. [42] combine multiple explainable reinforcement learning techniques to enhance transparency in adaptation decisions, aiming to foster user trust and facilitate debugging of control mechanisms. Camilli et al. [10] introduced a general framework for generating human-understandable explanations of both successful and (post-mortem) unsuccessful adaptation decisions, with a particular focus on robotic domains. Their work addresses the varying roles of humans in the adaptation loop, distinguishing between "out-of-the-loop" users—who do not intervene in the adaptation process—and "in-the-loop" users—who supervise and may influence adaptation outcomes [43–45]. Building on this perspective, existing methods integrate stochastic model checking to verify critical properties in complex, multi-stakeholder adaptation scenarios, and combine it with model-agnostic interpretable ML. This integration supports both designers and

human operators in understanding the interplay between environmental conditions, adaptation strategies, and potential points of failure [46, 47].

The *advantages* of these approaches include increased user trust and improved alignment with safety and regulatory requirements. By enhancing the interpretability of adaptation decisions, these techniques contribute to a better end-user experience and support more effective debugging by operators of autonomous systems. Some *limitations* remain. Many contributions lack systematic evaluation from the perspective of the human stakeholders who are expected to benefit from the explanations. Moreover, the integration of explainability with runtime constraints—such as time-critical adaptation steps and limited computational resources—has received limited attention, leaving open challenges in ensuring that explanations remain both timely and actionable in real-world scenarios.

Within this landscape, our unique contribution is the integration of diverse explainability mechanisms into the control loop of autonomous and adaptive systems. Rather than aiding human interpretability, this integration serves the following purposes: (i) guide and inform adaptation decisions, and (ii) enhance cost-effectiveness by transforming opaque black-box model predictions into interpretable white-box functions, thereby enabling more efficient and transparent optimization at runtime.

8 Conclusion

We present XDA-II, a novel approach that embeds model-agnostic interpretable ML within a MAPE-K control loop and integrates heterogeneous explanation sources to enhance the adaptation process, making it more efficient while preserving effectiveness. The key components include an offline pre-processor and an online white-box optimizer, working in tandem to leverage explainability and steer adaptation decisions. We conducted an empirical evaluation considering three open-source CPS benchmarks, each one in two versions, having increasing complexity, resulting in a total of six evaluation subjects. To show the cost-effectiveness of XDA-II, we compared the results with a number of baseline methods, including RS, the predecessor XDA, and mainstream methods based on metaheuristic optimizing search (NSGA-III and FITEST).

Results show that XDA-II generally achieves higher quality scores than other baseline methods, with significant improvements as the complexity of the evaluation subject increases, particularly in larger semantic spaces and with more CFs. The offline pre-processor cost varies with subject complexity, however, the online execution time and memory consumption are consistently lower than all other baseline methods (except RS), with statistically significant differences and large effect sizes across all study subjects.

We plan to explore the adoption of additional types of explanations for various, possibly conflicting, goals. As an example, we may adopt counterfactual explanations to drive automated repair processes of the adaptation control loop. We also plan to integrate lightweight online learning and dynamic explanation updating to extend the resilience of XDA-II under highly dynamic environments, for instance, in the presence of concept drifts.

References

[1] Sara Mahdavi-Hezavehi, Danny Weyns, Paris Avgeriou, Radu Calinescu, Raffaela Mirandola, and Diego Perez-Palacin. 2020. Uncertainty in self-adaptive systems: A research community perspective. *ACM Transactions on Autonomous and Adaptive Systems* 15, 4 (2020), Article 10, 1–36.

[2] Danny Weyns, Radu Calinescu, Raffaela Mirandola, and Kenji Tei. 2023. Towards a research agenda for understanding and managing uncertainty in self-adaptive systems. *ACM SIGSOFT Software Engineering Notes* 48, 4 (2023), 20–36.

[3] Michael Austin Langford and Betty H. C. Cheng. 2021. "Know What You Know": Predicting behavior for learning-enabled systems when facing uncertainty. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '21)*, 78–89. DOI: https://doi.org/10.1109/SEAMS51251.2021.00020

[4] Jia Li, Shiva Nejati, and Mehrdad Sabetzadeh. 2022. Learning self-adaptations for IoT networks: A genetic programming approach. arXiv:2205.04352. Retrieved from https://arxiv.org/abs/2205.04352

[5] Danny Weyns, Omid Gheibi, Federico Quin, and Jeroen Van Der Donckt. 2022. Deep learning for effective and efficient reduction of large adaptation spaces in self-adaptive systems. arXiv:2204.06254. Retrieved from https://arxiv.org/abs/2204.06254

[6] Omid Gheibi, Danny Weyns, and Federico Quin. 2021. Applying machine learning in self-adaptive systems: A systematic literature review 15, 3 (Aug. 2021), Article 9, 1–37. DOI: https://doi.org/10.1145/3469440

[7] V. N. Vapnik. 1999. An overview of statistical learning theory. *IEEE Transactions on Neural Networks* 10, 5 (1999), 988–999. DOI: https://doi.org/10.1109/72.788640

[8] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. Retrieved from https://www.sciencedirect.com/science/article/pii/S0893608014002135

[9] Omid Gheibi and Danny Weyns. 2022. Lifelong self-adaptation: Self-adaptation meets lifelong machine learning. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '22)*. ACM, New York, NY, 1–12. DOI: https://doi.org/10.1145/3524844.3528052

[10] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2023. XSA: Explainable self-adaptation. In *Proceedings of the International Conference on Automated Software Engineering (ASE '22)*. ACM.

[11] Francesco Renato Negri, Niccolo Nicolosi, Matteo Camilli, and Raffaela Mirandola. 2024. Explanation-driven self-adaptation using model-agnostic interpretable machine learning. In *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '24)*. ACM, New York, NY, 189–199. DOI: https://doi.org/10.1145/3643915.3644085

[12] Christoph Molnar. 2022. *Interpretable Machine Learning* (2nd ed.). Retrieved from https://christophm.github.io/interpretable-ml-book

[13] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. 2019. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. arXiv:1801.01489. Retrieved from https://arxiv.org/abs/1801.01489

[14] Naeem Esfahani and Sam Malek. 2013. Uncertainty in self-adaptive software systems. In *Proceedings of the Software Engineering for Self-Adaptive Systems II*. Springer, Berlin, 214–238. DOI: https://doi.org/10.1007/978-3-642-35813-5_9

[15] Shiva Nejati, Lev Sorokin, Damir Safin, Federico Formica, Mohammad Mahdi Mahboob, and Claudio Menghi. Reflections on surrogate-assisted search-based testing: A taxonomy and two replication studies based on industrial ADAS and Simulink models. *Information and Software Technology* 163 (2023), 107286. DOI: https://doi.org/10.1016/j.infsof.2023.107286. Retrieved from https://www.sciencedirect.com/science/article/pii/S0950584923001404

[16] Gabriel Moreno, Cody Kinneer, Ashutosh Pandey, and David Garlan. 2019. Dartsim: An exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems. In *Proceedings of the 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '19)*, 181–187. DOI: https://doi.org/10.1109/SEAMS.2019.00031

[17] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2022. Taming model uncertainty in self-adaptive systems using Bayesian model averaging. In *Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '22)*, ACM/IEEE, 25–35. DOI: https://doi.org/10.1145/3524844.3528056

[18] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. 2023. Enforcing resilience in cyber-physical systems via equilibrium verification at runtime. *ACM Transactions on Autonomous and Adaptive Systems* 18, 3 (Sep. 2023), Article 12, 1–32. DOI: https://doi.org/10.1145/3584364

[19] Clay Stevens and Hamid Bagheri. 2020. Reducing run-time adaptation space via analysis of possible utility bounds. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20)*. ACM, New York, NY, 1522–1534. DOI: https://doi.org/10.1145/3377811.3380365

[20] Kalyanmoy Deb and Himanshu Jain. 2014. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation* 18, 4 (2014), 577–601. DOI: https://doi.org/10.1109/TEVC.2013.2281535

[21] Sotiris B. Kotsiantis, Ioannis Zaharakis, and P. Pintelas. 2007. Supervised machine learning: A review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering* 160, 1 (2007), 3–24.

[22] Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence* 267 (2019), 1–38. DOI: https://doi.org/10.1016/j.artint.2018.07.007

[23] Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS '17)*. Curran Associates Inc., Red Hook, NY, 4768–4777.

[24] D. Randall Wilson and Tony R. Martinez. 1997. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research* 6, 1 (Jan. 1997), 1–34.

[25] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. ACM, New York, NY, 143–154. DOI: https://doi.org/10.1145/3238147.3238192

[26] Cody Kinneer, David Garlan, and Claire Le Goues. 2021. Information reuse and stochastic search: Managing uncertainty in self-* systems. *ACM Transactions on Autonomous and Adaptive Systems* 15, 1 (Feb. 2021), Article 3, 1–36. DOI: https://doi.org/10.1145/3440119

[27] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. ACM, New York, NY, 1–10. DOI: https://doi.org/10.1145/1985793.1985795

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

[29] András Vargha and Harold D. Delaney. 2000. A critique and improvement of the "cl" common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. Retrieved from http://www.jstor.org/stable/1165329

[30] Plamen P. Angelov, Eduardo A. Soares, Richard Jiang, Nicholas I. Arnold, and Peter M. Atkinson. 2021. Explainable artificial intelligence: An analytical review. *WIREs Data Mining and Knowledge Discovery* 11, 5 (2021), e1424.

[31] Luca Longo, Mario Brcic, Federico Cabitza, Jaesik Choi, Roberto Confalonieri, Javier Del Ser, Riccardo Guidotti, Yoichi Hayashi, Francisco Herrera, Andreas Holzinger, et al. 2024. Explainable artificial intelligence (XAI) 2.0: A manifesto of open challenges and interdisciplinary research directions. *Information Fusion* 106 (2024), 102301. DOI: https://doi.org/10.1016/j.inffus.2024.102301. Retrieved from https://www.sciencedirect.com/science/article/pii/S1566253524000794

[32] Markus Langer, Daniel Oster, Timo Speith, Holger Hermanns, Lena Kästner, Eva Schmidt, Andreas Sesing, and Kevin Baum. 2021. What do we want from explainable artificial intelligence (XAI)?—A stakeholder perspective on XAI and a conceptual model guiding interdisciplinary XAI research. *Artificial Intelligence* 296 (2021), 103473. DOI: https://doi.org/10.1016/j.artint.2021.103473. Retrieved from https://www.sciencedirect.com/science/article/pii/S0004370221000242

[33] Meike Nauta, Jan Trienes, Shreyasi Pathak, Elisa Nguyen, Michelle Peters, Yasmin Schmitt, Jörg Schlötterer, Maurice van Keulen, and Christin Seifert. 2023. From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable AI. *ACM Computing Surveys* 55, 13s (Jul. 2023), 1–42. DOI: https://doi.org/10.1145/3583558

[34] Chakkrit Kla Tantithamthavorn and Jirayus Jiarpakdee. 2021. Explainable AI for software engineering. *In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21)*, 1–2. DOI: https://doi.org/10.1109/ASE51524.2021.9678580

[35] Arut Prakash Kaleeswaran, Arne Nordmann, Thomas Vogel, and Lars Grunske. 2022. A systematic literature review on counterexample explanation. *Information and Software Technology* 145 (2022), 106800. DOI: https://doi.org/10.1016/j.infsof.2021.106800

[36] Javier Cámara, Mariana Silva, David Garlan, and Bradley R. Schmerl. 2021. Explaining architectural design tradeoff spaces: A machine learning approach. In *Proceedings of the 15th European Conference on Software Architecture (ECSA '21)*. Lecture Notes in Computer Science, Vol. 12857. Springer, 9–65. DOI: https://doi.org/10.1007/978-3-030-86044-8_4

[37] Larissa Chazette, Wasja Brunotte, and Timo Speith. 2021. Exploring explainability: A definition, a model, and a knowledge catalogue. In *Proceedings of the IEEE 29th International Requirements Engineering Conference (RE'21)*, Notre Dame, IN, USA,197–208. DOI: https://doi.org/10.1109/RE51729.2021.00025

[38] Mathias Blumreiter, Joel Greenyer, Francisco Javier Chiyah Garcia, Verena Klös, Maike Schwammberger, Christoph Sommer, Andreas Vogelsang, and Andreas Wortmann. 2019. Towards self-explainable cyber-physical systems. In *Proceedings of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C '19)*, 543–548. DOI: https://doi.org/10.1109/MODELS-C.2019.00084

[39] Alhassan Boner Diallo, Hiroyuki Nakagawa, and Tatsuhiro Tsuchiya. 2021. Adaptation space reduction using an explainable framework. In *Proceedings of the IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC '21)*. IEEE, 1653–1660. DOI: https://doi.org/10.1109/COMPSAC51774.2021.00247

[40] Nianyu Li, Sridhar Adepu, Eunsuk Kang, and David Garlan. 2020. Explanations for human-on-the-loop: A probabilistic model checking approach. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '20)*. ACM, New York, NY, 181–187. DOI: https://doi.org/10.1145/3387939.3391592

[41] Rebekka Wohlrab, Javier Cámara, David Garlan, and Bradley Schmerl. 2023. Explaining quality attribute tradeoffs in automated planning for self-adaptive systems. *Journal of Systems and Software*, 198 (2023), 111538.

[42] F. Feit, A. Metzger, and K. Pohl. 2022. Explaining online reinforcement learning decisions of self-adaptive systems. In *Proceedings of the IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS '22)*. IEEE Computer Society, Los Alamitos, CA, 51–60. DOI: https://doi.org/10.1109/ACSOS55765.2022.00023. Retrieved from https://doi.ieeecomputersociety.org/10.1109/ACSOS55765.2022.00023

[43] Rogério de Lemos. 2020. Human in the loop: What is the point of no return? In *Proceedings of the Sun Enterprise Authentication Mechanism (SEAMS '20)*. ACM, 165–166.
[44] Azad M. Madni and Carla C. Madni. 2018. Architectural framework for exploring adaptive human-machine teaming options in simulated dynamic environments. *Systems* 6, 4 (2018), 44.
[45] Nianyu Li, Javier Cámara, David Garlan, Bradley R. Schmerl, and Zhi Jin. 2021. Hey! Preparing humans to do tasks in self-adaptive systems. In *Proceedings of the Sun Enterprise Authentication Mechanism (SEAMS '21)*. IEEE, 48–58.
[46] Marcello M. Bersani, Matteo Camilli, Livia Lestingi, Raffaela Mirandola, Matteo Rossi, and Patrizia Scandurra. 2023. Towards better trust in human-machine teaming through explainable dependability. In *Proceedings of 2023 IEEE 20th International Conference on Software Architecture Companion*. IEEE, 86–90.
[47] Marcello M. Bersani, Matteo Camilli, Livia Lestingi, Raffaela Mirandola, and Matteo Rossi. 2023. Explainable human-machine teaming using model checking and interpretable machine learning. In *Proceedings of the IEEE/ACM 11th International Conference on Formal Methods in Software Engineering (FormaliSE '23)*, 18–28. DOI: https://doi.org/10.1109/FormaliSE58978.2023.00010

Received 9 November 2024; revised 27 March 2025; accepted 21 May 2025