Information and Software Technology 201 (2027) 108370

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

# Self-adaptation via joint global–local explanations for autonomous cyber–physical systems

Ehsan Elahi [a], Matteo Camilli [b,*], Raffaela Mirandola [a]

[a] Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
[b] Politecnico di Milano, Milano, Italy

ARTICLE INFO

Dataset link: Replication package (Original data)

*Keywords:*
Explainable AI
Cyber–physical systems
Self-adaptive systems

ABSTRACT

**Context:** Autonomous and self-adaptive systems increasingly rely on black-box predictive models to support runtime decision making in critical domains. However, the lack of transparency of such models enlarges the adaptation search space and increases optimization costs.
**Objective:** This paper addresses the problem of multi-objective optimization in opaque autonomous systems through cost-effective explanation-driven adaptation.
**Method:** We propose GLASS, a self-adaptation approach that integrates model-agnostic interpretable machine learning into a feedback loop. Building on our previous work, GLASS combines global explanations (Partial Dependence Plots) with local explanations (counterfactuals) to guide runtime adaptation decisions and reduce the search space toward goal-relevant configurations.
**Results:** An empirical evaluation on three cyber–physical systems benchmarks shows that GLASS achieves competitive or superior adaptation effectiveness while significantly reducing execution time compared to three state-of-the-art baselines, including up to two orders of magnitude lower overhead than black-box optimization.
**Conclusion:** GLASS enables transparent and efficient runtime self-adaptation by jointly leveraging global and local explanations to optimize adaptation search while preserving effectiveness.

## 1. Introduction

Modern software and Cyber–Physical Systems (CPSs) are increasingly deployed in dynamic and unpredictable environments—from cloud services managing fluctuating users' needs to autonomous driving navigating complex pathways. For these systems to operate without redeployment, they must adapt to manage uncertainties in the environment or within themselves. Self-Adaptive Systems (SASs) play a crucial role in managing uncertainties by adjusting their behavior to fulfill the desired goals [1,2]. SASs can modify their own behavior at runtime in response to changes in their environment, goals, or internal state. To engineer SAS, a primary approach is to incorporate an adaptation layer in its architecture. The adaptation layer usually integrates a MAPE-K feedback control loop, which *monitors*, *analyzes*, *plans*, and then *executes* adaptation decisions onto the managed layer. Pre-trained machine learning models are typically used in the feedback control loop to embed predictive capability and implement proactive decision-making [3]. Here, supervised learning approaches, such as neural networks and ensemble models, are typically used to build predictive models [4,5].

Although such integration has advantages, the opaque nature of machine learning models makes it difficult for humans to understand why a system adapts in a certain way, thus creating a barrier to trustworthiness [6]. Moreover, this lack of transparency introduces complexity and makes them computationally expensive, as they may require expensive iterative methods to explore a large adaptation space with (many) optimization goals [7]. This problem has led to the emerging domain of explanation-driven self-adaptation [3,8]. The approach makes use of *global* interpretable machine learning [9] to make adaptation decisions transparent. Such transparency typically yields efficient convergence toward adaptation decisions leveraging white-box optimization methods rather than expensive black-box optimization [3,10].

We introduce GLASS,[1] a novel self-adaptation approach that jointly leverages global and local explanations to guide runtime adaptation decisions. The approach extends our previous work [11], in which we introduced CSA-$\Phi$, a counterfactual-based adaptation mechanism exploiting local explanations to steer adaptation.

* Corresponding author.
*E-mail addresses:* ehsan.elahi@kit.edu (E. Elahi), matteo.camilli@polimi.it (M. Camilli), raffaela.mirandola@kit.edu (R. Mirandola).

[1] GLASS stands for Global–Local explAnations for Self-adaptive Systems.

https://doi.org/10.1016/j.infsof.2026.108370
Received 21 May 2026; Received in revised form 7 October 2026; Accepted 9 October 2026
Available online 21 October 2026
0950-5849/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

GLASS consists of two main stages: (*i*) an *offline preprocessing* phase and (*ii*) an *online white-box optimization* phase. During offline preprocessing, predictive models are trained for each system requirement (i.e., adaptation goal), and global explanations are computed in advance to support runtime decision making after deployment. During online optimization, the approach first generates counterfactual explanations [12] for the current operating condition and subsequently exploits Partial Dependence Plots (PDPs) [9] to refine and complement the resulting adaptation decisions. By combining local and global explanatory information, GLASS enables a more effective and targeted exploration of the adaptation space. Notice that GLASS treats explanations as machine-readable artifacts that constrain and guide the adaptation process. Rather than producing explanations primarily intended for human interpretation, we exploit transparency to improve the efficiency and effectiveness of runtime self-adaptation.

Compared to our prior work [11], GLASS integrates a global-explanation layer, namely PDPs (discussed in Section 3.2.2), within the MAPE-K feedback loop. While CSA-$\Phi$ relied exclusively on local counterfactual explanations to derive adaptation decisions, GLASS combines local and global explanatory information to guide adaptation more effectively. In particular, PDPs provide a principled, model-wide view of the relationship between controllable features and requirement satisfaction, enabling the system to steer adaptation toward globally favorable regions of the search space rather than relying solely on instance-specific adjustments.

We carry out an empirical evaluation of GLASS on three publicly available CPS benchmarks from different domains to assess the effectiveness, quality distance, and efficiency of adaptation decisions. The results show that GLASS consistently achieves competitive or superior requirement satisfaction likelihoods, typically ranging between 0.9 and 0.95 on the most complex benchmark, while also obtaining the highest success rates across the evaluated subjects. Moreover, GLASS significantly reduces adaptation overhead, achieving median execution times of approximately $\sim$0.02s, up to two orders of magnitude faster than the selected black-box optimization baseline.

Our main contributions can be summarized as follows:

- We introduce GLASS, an extension of our previous work [11], that integrates global and local explanations within a white-box optimization approach to enable efficient, transparent runtime adaptation.
- We conduct an empirical evaluation to assess the impact of global–local explanations in enhancing the efficiency and quality of adaptation decisions compared to selected baseline methods.
- We release a publicly available replication package, including sources and instructions to replicate our experiments.

The remainder of this paper is structured as follows. Section 2 introduces a running example from the robotics domain and formalizes the problem statement. Section 3 presents the necessary background concepts. Section 4 details the proposed approach, GLASS, while Section 5 reports the empirical evaluation. Section 6 discusses configuration parameters, assumptions, and limitations, while Section 7 discusses related work, and Section 8 concludes the paper.

## 2. Problem statement

This section outlines the problem description. To illustrate the main concepts, we consider a Rescue Robot (RR) self-adaptive system. RR operates in disaster scenarios such as earthquakes, hurricanes, and fires, where it is tasked with locating injured individuals [13]. The system comprises abstractions of physical interfaces, including sensors and actuators, and implements core rescue functionalities such as navigation, human and obstacle detection, and collision avoidance.

Operating in highly uncertain environments with fluctuating requirements and resource constraints, such as limited battery capacity and variable image resolution, RR must continuously adapt its behavior at runtime. To this end, its components are designed to be dynamically reconfigured. Reconfigurations are managed by an adaptation layer implementing a MAPE-K feedback control loop, which monitors the system and its environment, analyzes the need for adaptation, plans suitable reconfiguration actions, and executes the selected adaptation. The goal of this loop is to ensure that system-level requirements, expressed as probabilistic properties, are satisfied. Selected examples of such requirements include:

- $R_1$: "*The probability that RR maintains a safe protective distance from humans while moving must be greater than 0.9.*"
- $R_2$: "*The probability that RR avoids collisions with obstacles must be greater than 0.9.*"

**Table 1**
Semantic space of the rescue robot.

| Feature | Dimension | Type | Range |
|---|---|---|---|
| cruise speed | controllable | continuous | [0, 5] m/s |
| image resolution | controllable | categorical | {low, mid, high} |
| control responsiveness | controllable | continuous | [10, 50] Mbit/s |
| illuminance | controllable | continuous | [40, 120000] lux |
| power | observable | integer | [0, 100] % |
| smoke intensity | observable | categorical | {none, thin, thick} |
| obstacle size | observable | continuous | [0, 2] $m^3$ |
| obstacle distance | observable | continuous | [0, 10] m |
| firm obstacle | observable | boolean | Yes/No |

As discussed in Section 1, self-adaptive systems increasingly rely on black-box predictive models for runtime decision making. Following the common architecture adopted in machine-learning-enabled self-adaptive systems [14,15], GLASS assumes that the *Knowledge* component maintains pre-trained predictive models (e.g., neural network classifiers) that estimate the likelihood of requirement satisfaction under given operating conditions [7]. These operating conditions are characterized along two dimensions: *configuration* and *environment*. The configuration dimension consists of *controllable features*, which can be adjusted by the system to improve requirement satisfaction. In contrast, the environment dimension consists of *observable features*, which capture environmental factors that cannot be controlled but can be observed at runtime. Table 1 summarizes the set of relevant features for the RR example.

Given a specific value assignment to controllable and observable features, the black-box classifiers estimate the satisfaction likelihood of the system-level requirements (i.e., likelihood of achieving adaptation goals). Consequently, adaptation decisions can be derived from an optimization process that requires exploring a large and high-dimensional configuration space, which is computationally expensive. Furthermore, the lack of interpretability of the classifiers hinders reasoning about trade-offs among multiple, potentially conflicting objectives. As a result, adaptation mechanisms must balance computational cost and decision effectiveness to avoid the curse of dimensionality [10].

## 3. Background

This section provides the basic overview of supervised machine learning, followed by a discussion on interpretable machine learning, which together form the foundation of our proposed approach. For a detailed understanding of these concepts, we refer the reader to the resources in the bibliography.

### 3.1. Supervised machine learning

Supervised ML refers to the paradigm within AI in which an algorithm learns from the labeled data. Hence, this learned algorithm then maps the input data point (instance) to the target output based

on the data it learned [16,17]. The main goal of supervised ML is to predict the output label $y$ for the new unseen data (apart from the data it learned from). This form of learning, where we have defined labels as the output variable, is known as *classification*, and the algorithm that makes these predictions is known as the *classifier*. Classification models could be either binary (with two classes) or multiclass (more than two classes) and are built from learning the labeled data known as the *training set*, and are estimated on unseen data points known as the *test set*. The data point $x$ given to the classification function $\hat{f}$ to predict the output label $y_i$ is denoted as;

$$\hat{f}(x) = y_i \tag{1}$$

Supervised ML has many classifiers specific to different problems. Some models, such as decision trees and linear regression, are simple and inherently easy to understand their working, while others, such as neural networks, are complex and thus are black boxes.

### 3.2. Interpretable machine learning

The term *interpretable* refers to the level of human understanding of some phenomenon. In the same way, interpretable machine learning [9] is about the extraction of meaningful insights from the machine learning model to be well understood by humans. In this way, we get to know why a system made a certain decision [18]. As outlined by Miller [19], the terms *interpretable* and *explainable* will be used interchangeably in this paper. The interpretability could be either local or global. Global explanations provide the average behavior of the model, thereby offering a holistic view of the data distribution of the target output based on the features. In this way, not only are the important features determined, but the relationship between them is also understood.

In the following sections, we will discuss the interpretable machine learning techniques leveraged in this work to provide the reader with a high-level overview.

#### 3.2.1. Counterfactual explanation

Counterfactual explanation [12] is a model-agnostic interpretable machine learning technique that explains the outcomes of a given model in the form of "what-if" reasoning, such as, "what minimum changes are needed to flip the model's prediction to the desired output". Counterfactuals are local explanations, meaning that they work at the instance level. It is also possible to have more than one equally valid counterfactual for the given instance. This is known as the Rashomon effect [20].

Consider an instance (data point) $x$, that results in a predicted output $y$, the mathematical description of counterfactual explanation is;

$\hat{f}(x) = y$ (original output)

$\hat{f}(x') = y'$ (desired output)

$x'$ is obtained by slightly modifying the $x$, consequently, we have the desired output $y'$. One thing to note here is that counterfactuals $x'$ should be as similar as possible to the original instance $x$. In other words, the distance between the original instance and the counterfactual should be as minimal as possible.

**Example 3.1** (*Counterfactual Explanation*)**.** Consider the robot that is rescuing people in a disaster situation, having darkness and poor visibility in its environment. This makes it difficult for the robot to detect the obstacles correctly. To make it detect and classify the obstacles correctly, counterfactuals may suggest slightly changing the values of the features *image resolution* and *illuminance* from 25% to 30% and from 50% to 60%, respectively. Utilizing these insights from counterfactuals, the system requirement of classifying the obstacles correctly is satisfied, and thus the output is flipped from *False* (not satisfied) to *True* (satisfied).

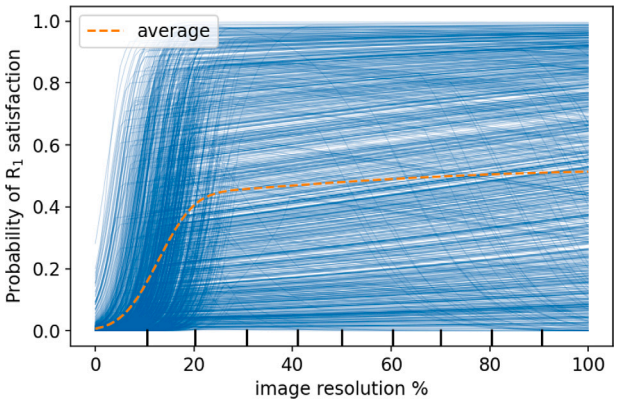



**Fig. 1.** Example of PDP.

#### 3.2.2. Partial dependence plot

PDP [9] is a global, model-agnostic, interpretable machine learning technique that describes the marginal effect of features (usually one or two) on the predicted output of a machine learning model. Thus, it can describe whether the relationship between the selected feature(s) and the target output is linear, monotonic, or complex. The PDP function is defined as

$$\hat{f}_s(\mathbf{x}_S) = E_{\mathbf{x}_C}\left[\hat{f}(\mathbf{x}_S, \mathbf{x}_C)\right] = \int \hat{f}(\mathbf{x}_S, \mathbf{x}_C)\, \mathrm{d}\mathbb{P}(\mathbf{x}_C). \tag{2}$$

where $\mathbf{x}_S$ identifies the features for which the partial dependence function should be plotted and $\mathbf{x}_C$ denotes the other features used in $\hat{f}$.

In our context, $\mathbf{x}_S$ contains controllable features, and PDP marginalizes the model output over the distribution of $\mathbf{x}_C$. Consequently, the relationship between selected controllable features and the target output is determined. In classification tasks, PDP displays the predicted probability for a given class label. Since PDP is a global explanation technique, it yields the global relationship of a feature with respect to the output variable.

**Example 3.2** (*PDP Explanation*)**.** Consider Fig. 1 showing an example from our illustrative RR system. We can see the selected feature *image resolution* (feature values are normalized to the [0–100] range) with average (the one in the dashed line) and individual conditional expectations (the continuous lines). The individual conditional expectation shows how the output variable (target variable) changes for each given sample in the distribution, that is, one line per sample. In the given example, the plot shows the relationship between the feature, *image resolution*, and the probability of satisfying the requirement $R_1$ of the robot. It is clearly visible that as the *image resolution* is increased from 10% to 20%, there is a sharp increase in the probability of satisfying $R_1$. In case of *image resolution* above 20%, the curve gets almost stable, with the maximum probability of satisfying $R_1$ is around 60%.

## 4. The glass approach

This section provides a detailed description of GLASS. We outline architectural aspects, followed by a description of its two main components: the offline preprocessor and the online white-box optimizer.

### 4.1. Architectural overview

GLASS is integrated into the MAPE-K loop, as illustrated in Fig. 2. The *Knowledge* component stores pre-trained predictive models $M_i$, each corresponding to a system requirement $R_i$. During operation, the *Monitor* component periodically collects the current operating conditions of the managed system, including both controllable and observable feature values. These conditions are then forwarded to the *Analyze*

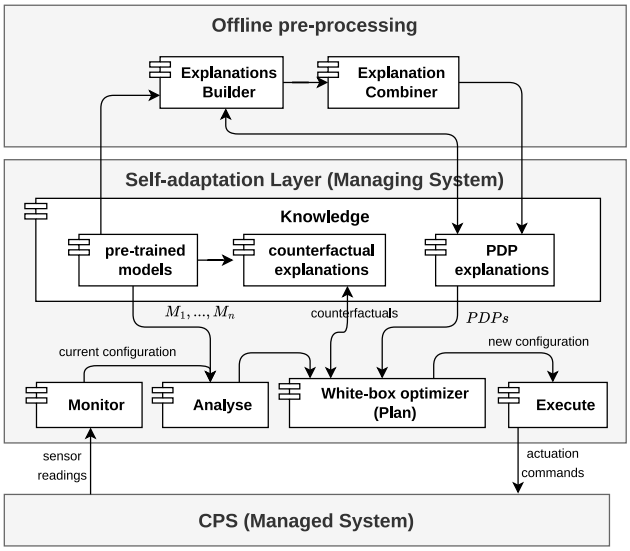



**Fig. 2.** Components of GLASS integrated into the MAPE-K loop.

component, which uses the corresponding model $M_i$ to estimate the probability of satisfying each requirement $R_i$. If at least one requirement is predicted to be violated (i.e., assigned a *false* class label), the *Analyze* component triggers the *Plan* component. The *Plan* component identifies candidate configurations of the controllable features aimed at increasing the likelihood of satisfying all requirements. The selected adaptation is then enacted by the *Execute* component, which updates the configuration of the managed system accordingly.

The offline preprocessor follows the approach originally introduced by Negri et al. [3] and includes two stages: an *explanation builder* and an *explanation combiner*. The explanation builder generates model-agnostic explanations, which are stored in the *Knowledge* component. The explanation combiner subsequently aggregates and structures these explanations into actionable insights that can be efficiently exploited at runtime. The online white-box optimizer leverages these precomputed explanations to guide adaptation decisions and reduce the search space of candidate configurations. As shown in Fig. 2, the online white-box optimization engine is embedded within the *Plan* component and operates after system deployment.

### 4.2. Offline pre-processor

In the offline phase, predictive models are trained for each system requirement and prepared for use during runtime adaptation. Algorithm 1 summarizes the steps performed in this phase. The training loop (lines 4–7) iterates over each requirement $R_i$ and trains one corresponding model $M_i$. The training labels for $M_i$ are binary: $y_i = 1$ if the requirement $R_i$ is satisfied under the given configuration, and $y_i = 0$ if $R_i$ was violated.

For simplicity, the presentation assumes a one-to-one correspondence between system requirements and predictive models (i.e., each requirement is associated with a dedicated binary classifier). More generally, GLASS supports many-to-many relationships between requirements and predictive models. For example, a single multi-label predictor may jointly estimate the satisfaction of multiple requirements.

As anticipated above, the offline pre-processor includes: a *build* stage and a *combination* stage. In the build stage, PDPs are computed for each requirement–feature pair. PDPs describe global relationships between feature values and the predicted probability of requirement satisfaction, thus reducing the adaptation search space. Since PDP computation is computationally expensive, performing this step offline avoids runtime overhead. Given the set of controllable features $CF$ and the trained predictive models $M_i$, Algorithm 1 computes $PDP_{ij}$, which captures the marginal effect of a controllable feature $CF_j$ on the satisfaction of requirement $req_i$. As illustrated in Fig. 1, a PDP represents both the average effect of a feature (dashed line) and its Individual Conditional Expectations (ICE) (solid lines).

**Table 2**
Example of PDP score computation.

| $x$ | $\text{ICE}_{R_1}$ | $\text{ICE}_{R_2}$ | Combined score |
|---|---|---|---|
| 10 | 0.30 | 0.50 | 0.15 |
| 20 | 0.60 | 0.50 | 0.30 |
| 30 | 0.70 | 0.40 | 0.28 |

The combination stage aggregates the computed PDPs to derive a score for each controllable feature assignment (line 15). Specifically, the score function combines the individual conditional expectations extracted from all predictive models by multiplying their corresponding values. A high score indicates that a given feature assignment is globally associated with an increased likelihood of satisfying multiple (potentially many[2]) system requirements simultaneously. Nevertheless, a high score does not necessarily guarantee the satisfaction of all requirements, since requirement satisfaction events may exhibit complex interdependencies. It is important to emphasize that such interdependencies among requirements are conceptually distinct from the feature-independence assumption underlying PDPs. While PDPs assume independence among features when estimating marginal effects, requirement interdependence concerns the relationships among system objectives and their simultaneous satisfaction.

The artifacts produced by Algorithm 1, including the trained models, the individual PDPs, and the combined scores ($SPDP$), are stored in the *Knowledge* component of the MAPE-K loop and are subsequently exploited by the online white-box optimizer.

**Example 4.1** (*PDP Score Computation*)**.** Consider two requirements, *R1*, *R2* and the controllable feature, *control responsiveness*, evaluated over $x \in \{10, 20, 30\}$. Suppose the ICE line associated with the neighbor instance identified in the current context yields the predicted probabilities as shown in Table 2. The combined score is obtained by multiplying, point-wise, the ICE values of the two requirement models at each grid point (line 15 of Algorithm 1). The maximal combined value occurs at $x = 20$, a score of 0.30, which is therefore selected as the PDP-derived optimal value for this feature, even though it is not the value that maximizes either requirement individually.

### 4.3. Online white-box optimizer

The online white-box optimizer implements a two-stage, explanation-driven strategy to generate refined adaptation decisions. This strategy produces adaptation configurations by combining complementary global and local explanations.

When triggered, the engine exploits counterfactual explanations to identify minimal changes to controllable features that would flip the predictive model's output toward the desired outcome. These counterfactuals provide local, instance-specific information and focus on efficient mitigation of predicted requirement violations. As a result, the first step produces a set of candidate adaptation configurations that represent (low-cost), locally optimal adjustments.

These candidate adaptations are further refined using global information available from PDPs. Since PDPs capture the average marginal

[2] The term *many* refers to optimization problems involving approximately 4 to 15 objectives, following the convention adopted in the many-objective optimization literature [21].

**Algorithm 1** Offline Pre-processing

**Input:** Controllable features $CF = \{CF_1, \ldots, CF_m\}$, System requirements $reqs$
**Output:** Trained models $\{M_1, \ldots, M_n\}$, Set of PDPs $PDP = \{PDP_{ij} \forall i = 1..n, j = 1..m\}$, set of summary PDPs $SPDP = \{SPDP_1, \ldots, SPDP_m\}$

Set $models \leftarrow \emptyset$
Set $PDP \leftarrow \emptyset$
Set $SPDP \leftarrow \emptyset$

**for** each $req$ in $reqs$ **do**
  $M_i \leftarrow train_model(req)$
  $models \leftarrow$ updateSet$(models, M_i)$
**end for**
▷ Build step
**for** $i \leftarrow 1$ to $n$ **do**
  **for** $j \leftarrow 1$ to $m$ **do**
    $PDP_{ij} \leftarrow$ computePDP$(M_i, CF_j)$
    $PDP \leftarrow$ updateSet$(PDP, PDP_{ij})$
  **end for**
**end for**
▷ Combination step
**for** $j \leftarrow 1$ to $m$ **do**
  $score_j \leftarrow$ computeScore$(PDP_{ij}, \ldots, PDP_{nj})$
  $SPDP \leftarrow$ updateSet$(SPDP, score_j)$
**end for**

**Algorithm 2** Counterfactual Hillclimbing

**Input:** data instance $x$, set of classifiers $M = \{M_1, ..., M_n\}$, confidence threshold $\tau$ (default 0.8), set of controllable features $CF = \{CF_1, ..., CF_m\}$, feature domains $\mathcal{D}$, $num_counterfactuals$ (default 3), $max_iterations$ (default 10)

**Output:** Set of adaptations $A$

$A \leftarrow \emptyset$
**for** $k = 1$ to $num_counterfactuals$ **do**
  $x^{(k)} \leftarrow$ counterfactual$(x, k)$
  $p \leftarrow$ predictProb$(M, x^{(k)})$
  iteration = 0
  **while** iteration < max_iterations ς $p < \tau$ **do**
    **for** each $cf \in CF$ **do**
      $\delta \leftarrow$ stepSize$(cf, \mathcal{D}_{cf})$
      $x^+ \leftarrow x^{(k)}[cf \leftarrow x^{(k)}_{cf} + \delta]$
      $x^- \leftarrow x^{(k)}[cf \leftarrow x^{(k)}_{cf} - \delta]$
      $x^* \leftarrow \arg\max_{x' \in \{x^+, x^-\}}$ predictProb$(M, x')$
      $p^* \leftarrow$ predictProb$(M, x^*)$
      **if** $p^* > p$ **then**
        $x^{(k)} \leftarrow x^*$
        $p \leftarrow p^*$
      **end if**
    **end for**
    iteration ++
    **if** $p \geq \tau$ **then**
      $A \leftarrow x^{(k)}$
      **break**
    **end if**
  **end while**
  **if** $A = \emptyset$ **then**
    $A \leftarrow x$ ▷ Fallback to original instance
  **end if**
**end for**
**return** $A$

effect of each feature on the model's predictions, they enable the engine to reason about model-wide behavior rather than isolated instances. By leveraging PDP information, the engine can prioritize candidate configurations that are not only locally effective but also globally "good" across a broader range of operating conditions. The refinement step relies on the precomputed PDPs generated during the offline phase and applies them to the counterfactual-based candidates. The rationale behind this combination is that counterfactual explanations efficiently identify minimal, instance-level changes, while PDPs provide a complementary global perspective that helps avoid locally optimal but globally suboptimal adaptations. Together, these two stages enable adaptation decisions that balance efficiency and effectiveness.

#### 4.3.1. Counterfactual hillclimbing

Algorithm 2 presents the high-level procedure used to generate counterfactual explanations for a given operating condition. The algorithm takes as input a set of classifiers $M$, a confidence threshold $\tau$, the set of controllable features $CF$, their corresponding domains $\mathcal{D}$, the number of counterfactuals, and the maximum number of iterations. Starting from the original data instance $x$, it initializes a candidate counterfactual $x^{(k)}$ and computes its predicted satisfaction probability using the available predictive models (lines 3–4).

For each controllable feature, the algorithm computes a feasible perturbation step $\delta$ according to the feature domain and generates two neighboring candidates, $x^+$ and $x^-$, by perturbing the feature value in both increasing and decreasing directions (lines 7–9). This bidirectional exploration promotes diversity among candidate counterfactuals while ensuring that generated adaptations remain within admissible feature-domain constraints. The perturbation step is calculated as a fixed fraction of the controllable feature's domain rather than being manually specified for each feature. Specifically, if a feature has a domain $[lower, upper]$, the perturbation step is calculated as $(upper - lower) \cdot \alpha$, where $\alpha$ is a configurable factor.[3] This formulation normalizes the perturbation across features with different value ranges. The perturbation strategy is designed to balance local exploration granularity with search efficiency, thus enabling meaningful neighborhood sampling without introducing excessive computational overhead.

Among the generated neighbors, the algorithm opportunistically selects the candidate that satisfies the feature-domain constraints and has a higher predicted probability of satisfying requirements. If the resulting probability satisfies the confidence threshold $\tau$, the candidate is added to the set of possible adaptations $A$. Otherwise, the algorithm iteratively expands the local neighborhood exploration following a *hill-climbing* strategy [22], progressively selecting the neighboring candidate that increases the predicted probability of requirement satisfaction until either a satisfactory candidate is identified or the stopping criterion is reached.

The proposed strategy treats controllable features independently while explicitly respecting domain constraints. This localized exploration process encourages the generation of minimal, instance-specific adaptations that satisfy counterfactual conditions while avoiding unnecessary deviations from the current configuration. Finally, if no valid counterfactual satisfying the confidence criterion can be identified, the algorithm falls back to the original operating condition (line 26), indicating that no safe adaptation could be derived within the explored neighborhood.

**Example 4.2** (*Counterfactual Hillclimbing*)**.** Consider the RR illustrative example. Suppose the current operating condition corresponds to a cruise speed of 4.5 m/s, for which the predictive model estimates a low probability of satisfying the obstacle detection requirement. The algorithm generates counterfactuals, for instance, suggesting that cruise speed reduction below 4.2 m/s may flip the prediction.

Among these candidates, the algorithm may identify configurations in which controllable features, including cruise speed, are perturbed. Some candidates exhibit cruise speeds below 4.2, m/s, leading to an increased predicted probability of successful obstacle detection. If the candidates satisfy the feature-domain constraints and achieve a confidence level greater than the threshold $\tau$ (e.g., 0.8), the algorithm selects one of them as the preferred adaptation, prioritizing the candidate with the highest confidence level.

#### 4.3.2. PDP-guided refinement

Algorithm 3 describes the PDP-guided refinement procedure applied to the candidate adaptations generated by the counterfactual stage. The

[3] By default, GLASS adopts $\alpha = 0.25$ (25% of the feature range), as preliminary sensitivity analysis indicated good balance between local search granularity and computational cost. Results for additional values of $\alpha$ are included in the replication package.

**Algorithm 3** PDP-Guided Adaptation Refinement

**Input:** Set of initial adaptations $A = \{a_1, \ldots, a_k\}$, current system state $s$, set of classifiers $M = \{M_1, \ldots, M_n\}$, confidence threshold $\tau$ (default 0.8), $max_possibilities$ (default 10000) set of controllable features $CF = \{CF_1, \ldots, CF_m\}$, set of external features $OF = \{OF_1, \ldots, OF_l\}$, set of $SPDP = \{SPDP_1, \ldots, SPDP_m\}$
**Output:** Set of refined adaptations $\{\hat{A} = \{\hat{A}_1, \ldots, \hat{A}_m\}\}$
**foreach** $adaptation \in A$
1: $full_vector \leftarrow combineFeatures(adaptation, s_{OF})$ ▷ Step 1: Find similar context
2: $neighbor_idx \leftarrow knn.findNeighbor(full_vector)$
3: $maximals \leftarrow \emptyset$ ▷ Step 2: Extract info from PDPs
4: **for** $i \leftarrow 1$ to $CF$ **do**
5: $maximals[i] \leftarrow computeMaximals(SPDP_i, neighbor_idx)$
6: **end for**
7: $n_possibilities \leftarrow total_combinations(maximals)$ ▷ Step 3: Manage combinatorial explosion
8: **while** $n_possibilities > max_possibilities$ **do**
9: $idx \leftarrow \arg\max_i |maximals[i]|$
10: $maximals[idx] \leftarrow downsampling(maximals[idx])$
11: $n_possibilities \leftarrow total_combinations(maximals)$
12: **end while**
13: $configurations \leftarrow computeCartesianProduct(maximals)$ ▷ Step 4: Generate and evaluate candidates
14: $confidence \leftarrow evaluateConfidence(M, configurations)$
15: **if** $\exists c \in confidence : c \geq \tau$ **then**
16: $\hat{A} \leftarrow \hat{A} \cup \{(configuration, conf) \mid conf \geq \tau\}$
17: **continue**
18: **end if**
**end foreach**
19: **return** $\hat{A}$

algorithm takes as input the set of candidate adaptations $A$ (produced by Algorithm 2), the set of classifiers $M$, the controllable features $CF$, and the precomputed PDP and summary PDP artifacts.

To ensure that refinement is contextual rather than generic, the algorithm first identifies a system configuration that is similar to the candidate adaptation under consideration (lines 1–2). This step grounds PDP guidance in the local operating context of the system. Then, for each controllable feature, the algorithm extracts the feature values corresponding to regions of the PDP curve that maximize the predicted probability of requirement satisfaction. In this way, exploration of candidate adaptations is guided by global model knowledge rather than random perturbations (lines 3–6).

Notice that for each controllable feature, the maximal values are extracted directly from the corresponding precomputed PDP. At runtime, GLASS first identifies the PDP curve associated with the operating condition most similar to the current state by selecting its nearest neighbor. It then scans all sampled feature values on that PDP and determines the maximum predicted probability. Every feature value whose PDP prediction equals this maximum is included in the maximal set. As a result, multiple maximal values may be returned when the PDP exhibits a plateau rather than a unique optimum.

Given the PDP-derived optimal regions for all controllable features, the algorithm computes the space of possible configuration combinations. Since the space of combinations grows combinatorially, the algorithm applies an iterative downsampling strategy to progressively reduce the number of candidate configurations while preserving diversity in the exploration (lines 7–12). At each iteration, it selects the controllable feature with the largest set of maximal values and halves this set by retaining every second value, so the remaining values still span the whole PDP-optimal region. The total number of combinations is then recomputed, and the process repeats until the number of candidate configurations falls within the computational budget. The remaining candidate configurations are generated via a Cartesian product over the selected feature values and subsequently evaluated using the predictive models.

Each candidate configuration is assessed against the confidence threshold $\tau$ across all requirements. Configurations that meet or exceed the threshold are retained, and the candidate with the highest overall confidence is selected as the refined adaptation decision (lines 13–18).

**Example 4.3** (*PDP-Guided Refinement*)**.** Consider the RR example. For clarity, we focus on a single controllable feature: *image resolution*. Suppose the predictive model estimates that the current system configuration fails to satisfy the given safety requirement $R_2$ (see Section 2).

After creating the candidate adaptations, the algorithm considers the PDP for the feature *image resolution*. As illustrated in Fig. 1, the PDP reveals a steep increase in the probability of requirement satisfaction as image resolution increases up to approximately 20 units, followed by a plateau beyond that point. Based on this, the algorithm samples a set of plausible image resolution values around the region that maximizes the predicted satisfaction probability. These values define the candidate space for refinement. Since combining plausible values across multiple controllable features may lead to a combinatorial explosion, the algorithm applies iterative downsampling to reduce the candidate space while preserving diversity in the exploration.

The remaining candidate configurations are then generated and evaluated using the predictive models. Configurations whose predicted confidence meets or exceeds the threshold $\tau$ (e.g., $\tau = 0.8$) are retained, and the configuration with the highest confidence is selected as the refined adaptation.

**Table 3**
Feature values that map to global optima (according to PDPs).

| Controllable feature | PDP maximal values |
|---|---|
| Cruise Speed | [41, 42, 43, … , 56] |
| Camera Quality | [85, 86, 87, … , 95] |
| Control Responsiveness | {71, 72, … , 80} |

**Example 4.4** (*Downsampling*)**.** Consider the RR example again. For simplicity, assume that all controllable features range within $[0, 100]$. Suppose that the counterfactual stage produces a candidate adaptation with the following feature values: *cruise speed* = 35, *camera quality* = 86, *control responsiveness* = 77, and *illuminance* = 69. PDP curves corresponding to operating conditions similar to this assignment are then selected, and for each controllable feature, sets of globally optimal values are extracted by analyzing these curves (e.g., see Table 3).

Using these PDP-derived maximal sets, Algorithm 3 initially computes the total number of possible configuration combinations via a Cartesian product. In this example, the number of combinations is $16 \times 11 \times 15 \times 10 = 26400$, which exceeds the predefined computational budget (e.g., 10k). To mitigate the resulting combinatorial explosion, the algorithm applies an iterative downsampling strategy.

In the first iteration, the controllable feature with the largest set of global optima, *cruise speed*, with 16 values, is selected, and a subset of its values is sampled, reducing the set to 8 representative values. The number of candidate configurations is then recomputed as $8 \times 11 \times 15 \times 10 = 13200$, which still exceeds the budget. The algorithm therefore proceeds to downsample the feature with the next largest maximal set, *control responsiveness*, which initially contains 15 values. After downsampling this set, the total number of candidate configurations is reduced to $8 \times 11 \times 8 \times 10 = 7040$, which falls below the budget. At this point, no further downsampling is required, and the remaining candidate configurations are evaluated using the predictive models.

## 5. Evaluation

In this section, we discuss the empirical evaluation of GLASS using three selected CPS benchmarks as study subjects. We consider the following research questions:

- **RQ1:** How effective is GLASS compared to the baseline methods in terms of satisfaction likelihood and success rate?
- **RQ2:** What is the quality distance of adaptation decisions made by GLASS compared to baseline methods?
- **RQ3:** What is the computational cost of GLASS compared to the baseline methods?
- **RQ4:** How sensitive is GLASS to variations in its hyperparameters?

**Table 4**
Structural characteristics of our study subjects.

| Subject | Controllable features | Observable features | Requirements |
|---|---|---|---|
| RR | 4 | 5 | 4 |
| ADAS | 1 | 5 | 3 |
| UAV | 3 | 4 | 12 |

### 5.1. Design of the evaluation

In the following, we describe the design of the evaluation, including evaluation subjects, baseline methods, statistical tests, and testbed.

#### 5.1.1. Evaluation subjects

We conduct an empirical evaluation using three publicly available CPS benchmarks as evaluation subjects. These benchmarks differ in the size of their semantic spaces and in the number and nature of the requirements they must satisfy. Table 4 summarizes the main characteristics of the CPS benchmarks considered in this study. The benchmarks are selected to cover different levels of structural complexity, captured by the number of controllable and observable features, as well as varying numbers of requirements. This diversity enables a systematic analysis of how increasing system complexity affects the efficiency and effectiveness of adaptation decisions.

The *ADAS subject* [23] represents an autonomous driving system equipped with an automated emergency braking component. The system operates with minimal human intervention and relies on runtime adaptation planning to satisfy safety requirements, such as maintaining a safe distance from pedestrians and obstacles. Compared to the other evaluation subjects, ADAS constitutes a relatively simple edge-case scenario, as its adaptation space includes only a single controllable feature. This characteristic makes it particularly suitable for assessing the behavior of GLASS in low-dimensional adaptation settings, where search-space optimization is less challenging. The *RR subject* [24] (see Section 2) models an autonomous rescue robot operating in catastrophic scenarios such as earthquakes or fires. The system is equipped with sensors and actuators to support navigation and rescue operations while ensuring safety-critical requirements (e.g., $R_1$, $R_2$ in Section 2). The *UAV subject* [25] models a team of unmanned aerial vehicles operating in a hostile environment to execute a surveillance mission. To accomplish the mission, the UAVs must continuously adapt their flight altitude to balance conflicting objectives, namely maximizing target detection effectiveness while minimizing exposure to potential threats and damage. Among the selected evaluation subjects, UAV represents the most complex adaptation scenario, as it involves the largest number of system-level requirements (12 in total) together with a multi-dimensional adaptation space. Specifically, its semantic space comprises seven features, three of which are controllable.

#### 5.1.2. Baseline methods

We compare GLASS against a set of baseline approaches selected for their relevance to the problem setting and their representativeness of different adaptation strategies. These baselines provide reference points for evaluating the efficiency and effectiveness of GLASS and include both black-box and white-box approaches, as follows:

- NSGA-III: a many-objective metaheuristic optimization algorithm [21]. We use it to explore the configuration search space to identify adaptation decisions that maximize the likelihood of satisfying system-level requirements simultaneously. We select NSGA-III as a baseline to assess the computational cost and effectiveness of GLASS with respect to mainstream black-box optimization methods. The requirement is considered to be satisfied if the predicted confidence achieves the predefined threshold ($\tau = 0.8$). The implementation is provided by the pymoo framework [26] with Das–Dennis reference directions ($n_partitions = 12$), from which the population size is automatically determined. Optimization terminates after at most 100 generations using the default multi-objective convergence criterion. For reproducibility, a fixed random seed is adopted.
- XDA: an existing explanation-driven white-box adaptation approach [3] that leverages global model explanations in the form of Partial Dependence Plots (PDPs) to make adaptation decisions transparent. PDPs are computed offline to mitigate runtime overhead, while the online adaptation phase exploits these precomputed explanations to guide decision-making efficiently. At runtime, the current operating condition is first matched to its eight nearest neighbors to identify the most relevant context. Candidate adaptations are then generated from the PDP-derived optimal feature values. To prevent a combinatorial explosion, the number of candidate configurations is bounded to 10,000 through iterative downsampling. Candidate adaptations are evaluated using the predictive models, and only those satisfying the confidence threshold ($\tau = 0.8$) are considered valid. The final adaptation is selected according to the optimization score.
- CSA-$\Phi$: our prior approach [11] that integrates local counterfactual explanations into the *Plan* component of the MAPE-K loop to steer adaptation decisions at runtime. Consistent with the other baseline methods, a requirement is considered to be satisfied if the predicted confidence achieves the predefined threshold ($\tau = 0.8$). It serves as a direct point of comparison to evaluate the added value of combining local and global explanations in GLASS.

#### 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 GLASS and all the selected baseline methods. According to the guideline introduced by Arcuri and Briand [27], we apply Mann–Whitney U tests [28] to evaluate the statistical significance of the results.

#### 5.1.4. Testbed

All experiments were carried out on a commodity laptop running the Windows 11 operating system (Version 10.0, Build 22631). The laptop is equipped with an Intel Core i5-10300H processor (4 cores, 8 CPUs, 2.50 GHz) and 8 GB of RAM.

### 5.2. Results

#### 5.2.1. Effectiveness (RQ1)

To address RQ1, we run GLASS and the selected baseline methods using the evaluation subjects listed in Table 4. Each initial configuration is the operating condition that yields a violation of one or more requirements. We execute 200 runs from different initial configurations to compute and compare the effectiveness of adaptation decisions. To measure the effectiveness, we consider two metrics: *satisfaction likelihood* and *success rate*.

Notice that we adopt the same evaluation metrics as those used in the original XDA study [3], which we briefly summarize below for completeness.

**Metric 1** (*Satisfaction Likelihood*)**.** Given the requirement $R_i$, satisfaction likelihood is the measure of probability predicted by the black-box model $M_i$ for the class label to be *true*, having an adaptation configuration that represents assignments of new values to controllable features. Mathematically, it can be expressed as follows.

$$Likelihood(R_i) = P_{M_i}\left(y = \text{true} \mid (\hat{CF}, \hat{OF})\right) \tag{3}$$

where $\hat{CF}$ represents the adaptation configuration applied to controllable features and $\hat{OF}$ is the current state of the observable features.

**Table 5**
Statistical test for the satisfaction likelihood in the RR and ADAS subjects. Statistically significant values ($p < 0.05$) are highlighted.

| Req. | RR | | | ADAS | | |
|---|---|---|---|---|---|---|
| | NSGA-III | XDA | CSA-$\Phi$ | NSGA-III | XDA | CSA-$\Phi$ |
| $R_1$ | 8.9e−06 | 4.5e−05 | 5.6e−03 | 0.62 | 0.50 | 0.33 |
| $R_2$ | 2.1e−22 | 1.4e−23 | 2.4e−09 | 2.5e−21 | 3.1e−19 | 4.2e−25 |
| $R_3$ | 7.5e−17 | 6.8e−12 | 5.8e−06 | 0.30 | 0.41 | 0.35 |
| $R_4$ | 2.8e−12 | 6.4e−11 | 4.7e−07 | - | - | - |

**Table 6**
Statistical test for the satisfaction likelihood in the UAV subject. Statistically significant values ($p < 0.05$) are highlighted.

| Req. | NSGA-III | XDA | CSA-$\Phi$ |
|---|---|---|---|
| $R_1$ | 0.2 | 0.12 | 0.12 |
| $R_2$ | 1.1e−22 | 1.2e−02 | 1.2e−03 |
| $R_3$ | 4.3e−24 | 2.6e−16 | 3.2e−17 |
| $R_4$ | 1.4e−21 | 4.8e−02 | 2.7e−03 |
| $R_5$ | 3.6e−14 | 3.7e−03 | 1.7e−04 |
| $R_6$ | 2.6e−17 | 1.7e−02 | 1.6e−02 |
| $R_7$ | 6.2e−26 | 2.6e−03 | 9.5e−04 |
| $R_8$ | 1.9e−17 | 7.3e−04 | 7.3e−04 |
| $R_9$ | 6.4e−07 | 0.32 | 0.23 |
| $R_{10}$ | 2.2e−11 | 5.8e−04 | 5.4e−04 |
| $R_{11}$ | 3.7e−21 | 6.1e−23 | 6.4e−14 |
| $R_{12}$ | 7.4e−22 | 2.7e−13 | 4.6e−12 |

**Metric 2** (*Success Rate*)**.** Given a requirement $R_i$, success rate is defined as the number of instances in $X$ that satisfy that requirement $R_i$ under the given operating conditions ($\hat{CF}'\hat{OF}$), such that the predicted probability for the class label to be *true* is greater than the threshold $\tau$.[4] In other words, the success rate is defined as the proportion of instances in $X$ that are successful. Mathematically, it is given as follows.

$$SuccessRate(R_i) = \frac{n_{\text{success}}(R_i)}{|X|} \tag{4}$$

where $n_{\text{success}}(R_i)$ indicates the number of successful cases in which the requirement $R_i$ is satisfied, while $|X|$ represents the total number of evaluated instances $X$ (i.e., 200 instances per evaluation subject).

**Results.** Fig. 3 compares the distributions of requirement satisfaction likelihood achieved by GLASS and the baseline approaches across the three CPS benchmarks. To assess whether the observed differences are statistically significant, we apply the Mann–Whitney U test; the resulting $p$-values are reported in Tables 5 and 6, with statistically significant differences ($p < 0.05$) highlighted.

When the adaptation problem is relatively simple, as in the ADAS benchmark, which has a single controllable feature, all approaches exhibit comparable performance. Notice that the purpose of including ADAS as an evaluation subject is to validate GLASS in a controlled, low-dimensional setting, thus offering a range of controllable features ranging from one to multiple. To assess the search-space reduction under multi-dimensional adaptation, RR and UAV evaluation subjects are leveraged, which have four and three controllable features, respectively, as stated in Table 4.

For the RR benchmark, GLASS consistently achieves higher median satisfaction likelihoods and tighter distributions across all requirements compared to the baselines. For the UAV benchmark, which involves 3 controllable features and 12 requirements, GLASS attains median satisfaction likelihoods that are comparable to or slightly higher than those of the baselines for most requirements, typically ranging between 0.9 and 0.95. These results indicate that GLASS maintains strong performance as the dimensionality of the adaptation space and the number of requirements increase.

As anticipated above, we also evaluate the effectiveness of GLASS using the *success rate* metric across all requirements for each CPS benchmark. Notice that the success rate is an aggregate metric computed as the proportion of successful adaptation runs. Unlike the satisfaction likelihood, which provides one continuous observation per run and therefore naturally lend themselves to distribution-based statistical analysis, the success rate is a single aggregate proportion derived from these outcomes.

As shown in Fig. 4, GLASS exhibits similar or superior success rates across all subjects when compared to the baseline methods. This confirms the robustness of GLASS in maximizing the number of simultaneously satisfied requirements across CPS domains with varying levels of complexity.

> **RQ1 summary.** The results show that GLASS maintains competitive or superior satisfaction likelihood and success rates across all benchmarks, with statistically significant gains in more complex scenarios. This suggests that GLASS effectively supports multi-requirement adaptation as system complexity grows.

### 5.2.2. Quality distance of adaptation decisions (RQ2)

We answer RQ2 by using the same setting as RQ1 but computing the *quality distance* for all methods under comparison and all selected subjects in Table 4. The score is defined as follows.

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

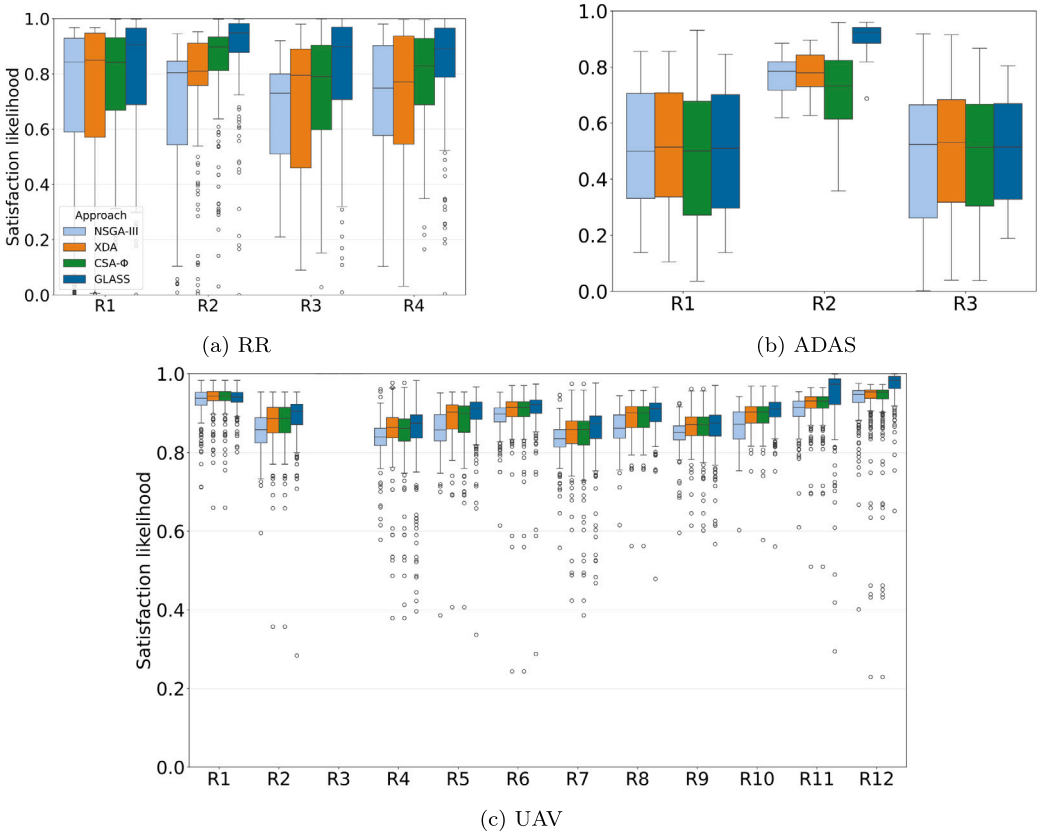
The ideal operating condition $I$ is domain-specific and is defined according to the optimization direction of each controllable feature. Specifically, features to be maximized take their upper-domain bound as the ideal value, whereas features to be minimized take their lower-domain bound. For example, in the RR subject, the ideal operating condition is $I = [100, 0, 0, 0]$. This configuration reflects the objective of maximizing the cruise speed (upper bound of 100), while minimizing image resolution, illuminance, and control responsiveness (lower bound of 0) to reduce energy consumption. The quality distance $d_{\text{norm}}$ quantifies the proximity of post-adaptation operating conditions with respect to ideal conditions (the smaller the distance, the better). We use normalization to adjust scale differences in the controllable features, ensuring that each feature contributes equally to the distance calculation regardless of domain size.

**Results.** The distributions of quality distances obtained by GLASS and the baseline approaches across the three subjects are shown in Fig. 5. Table 7 reports the results of the statistical significance tests. As lower quality distances indicate configurations closer to the ideal operating conditions, approaches with smaller medians are preferred.

For the RR subject, GLASS achieves a lower median quality distance than XDA and NSGA-III. CSA-$\Phi$ achieves the lowest median quality distance overall.

In the ADAS subject, GLASS yields a median quality distance comparable to those of XDA and NSGA-III but with a tighter distribution, suggesting more stable behavior across runs. In contrast, XDA and NSGA-III show similar medians but larger dispersion, with XDA exhibiting several high-value outliers. CSA-$\Phi$ displays the largest variability, with both extremely high and low scores. This behavior can be attributed to its exclusive reliance on counterfactual explanations without refinement using global information.

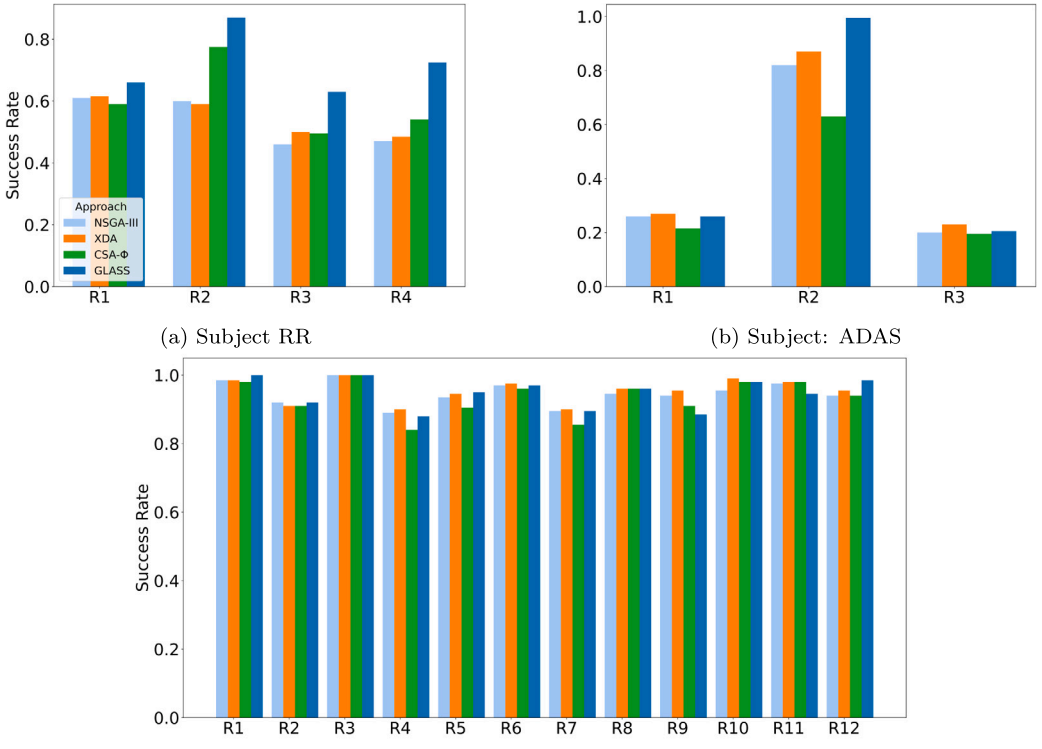
[4] We set the confidence threshold $\tau = 0.8$, ensuring consistency with the configuration used by all baseline methods.




(a) RR (b) ADAS

(c) UAV

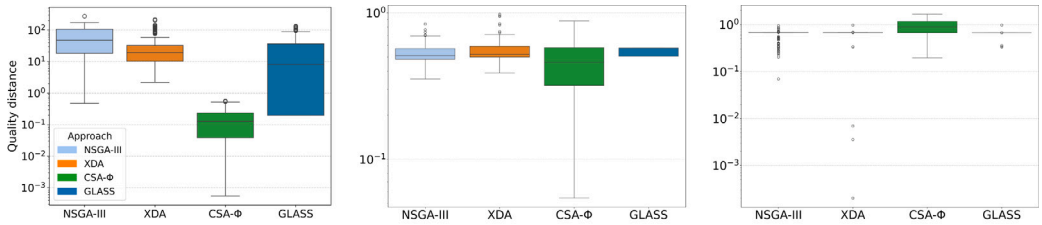
**Fig. 3.** Distribution of satisfaction likelihood across requirements (higher is better).




(a) Subject RR (b) Subject: ADAS

(c) Subject UAV

**Fig. 4.** Success rate (higher is better).




(a) Subject RR (b) Subject ADAS (c) Subject UAV

**Fig. 5.** Quality distance (lower is better).

**Table 7**
Mann–Whitney U test for statistical significance of the quality distance. Significant values ($p < 0.05$) are highlighted.

| Comparison | | RR | ADAS | UAV |
|---|---|---|---|---|
| A | B | ($p$-value) | ($p$-value) | ($p$-value) |
| GLASS | NSGA-III | 4.9e−31 | 1.1e−05 | 2.8e−10 |
| GLASS | XDA | 4.6e−46 | 4.2e-09 | 2.2e-22 |
| GLASS | CSA-$\Phi$ | 0.2 | 0.5 | 0.4 |

**Table 8**
Mann–Whitney U test for statistical significance of the execution time. Significant values ($p < 0.05$) are highlighted.

| Comparison | | RR | ADAS | UAV |
|---|---|---|---|---|
| A | B | ($p$-value) | ($p$-value) | ($p$-value) |
| GLASS | NSGA-III | 4.8e−36 | 4.5e−32 | 4.7e−38 |
| GLASS | XDA | 8.8e−12 | 5.8e−23 | 1.2e−14 |
| GLASS | CSA | 0.5 | 8.5e−44 | 0.5 |

For the UAV benchmark, GLASS achieves a median quality distance comparable to those of XDA and NSGA-III. Both NSGA-III and XDA exhibit a larger spread and multiple low-value outliers, indicating less robust performance across runs despite achieving competitive median values.

**RQ2 summary.** The results indicate that GLASS achieves quality distances that are comparable to or better than the baseline methods while exhibiting greater stability across runs. This suggests that GLASS consistently produces post-adaptation configurations closer to the ideal operating conditions, particularly in more complex settings.

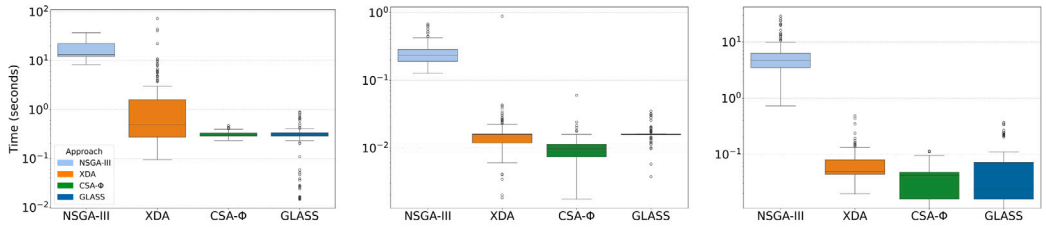
### 5.2.3. Computational cost (RQ3)

We address RQ3 using the same setup as RQ1 and RQ2, but we compute the computational cost in terms of wall-clock execution time taken by the plan component of the MAPE-K loop. Specifically, we compute the time taken by each of the initial configurations to adapt to the final configuration, that is, the value assignment to controllable features in the feature space. The execution time of GLASS is compared with the selected baselines, using the subjects in Table 4. The reported execution times quantify only the computational overhead of the planning algorithm under controlled experimental conditions. They do not account for deployment-specific factors (e.g., hardware resource constraints, sensor acquisition and processing, communication latency, operating system scheduling, or actuator execution delays). All approaches are evaluated under the same experimental conditions to compare relative computational cost.

**Results.** Fig. 6 compares the computational cost of GLASS and the baseline approaches in terms of execution time across all evaluation subjects. Table 8 reports the results of the statistical significance tests. For the RR subject (largest feature space), GLASS achieves significantly lower execution times than NSGA-III and XDA, while performing comparably to CSA-$\Phi$. The median execution times of GLASS and CSA-$\Phi$ are 0.30s and 0.29s, respectively, indicating a negligible difference. It is shown in Fig. 6(a). In the ADAS subject, GLASS exhibits a median execution time of approximately 0.015s, representing a substantial improvement over the baseline methods, particularly NSGA-III, whose median execution time is almost an order of magnitude higher. CSA-$\Phi$ exhibits a slightly lower median execution time (around 0.02s). In Fig. 6(b), all white-box approaches (XDA, CSA-$\Phi$ and GLASS) run in the order of $10^{-2}$ s. GLASS nevertheless remains about an order of magnitude faster than NSGA-III. For the UAV subject (Fig. 6(c)), GLASS achieves the lowest median execution time (approximately 0.03s) on our experimental setup, which is two orders of magnitude faster than NSGA-III, as well as statistically significantly lower than those of XDA and NSGA-III (see Table 8).

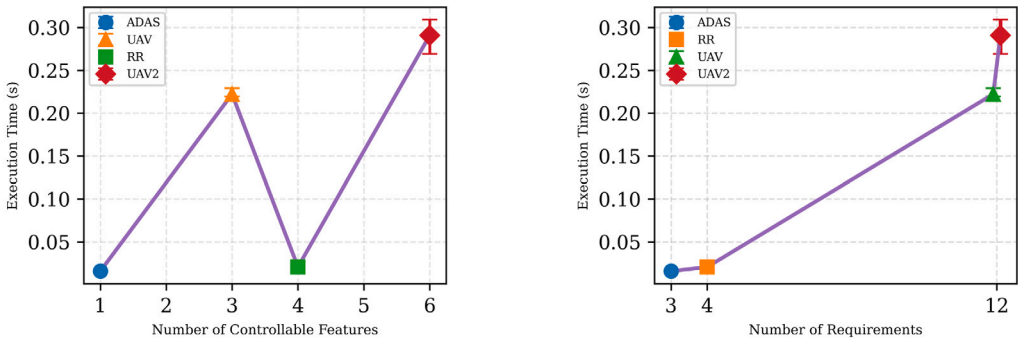
We assess the scalability of GLASS by analyzing the computational cost (i.e., time taken in seconds) of the plan component of the MAPE-K loop with respect to two independent dimensions of problem complexity: (i) the number of system requirements and (ii) the number of controllable features. Fig. 7(a) shows the execution time with respect to the number of controllable features, while Fig. 7(b) shows the execution time as the number of system requirements increases. Fig. 7(a) illustrates the effect of the adaptation space given in the semantic space. As the number of controllable features increases from one (in the case of ADAS) to six (in the case of UAV2[5]), the computational cost also increases. The increase is moderate rather than exponential, indicating that the search strategy effectively limits unnecessary exploration. Considering the effect of the number of requirements (Fig. 7(b)), the execution time generally increases as additional requirements must be considered during adaptation planning. ADAS, which consists of only three requirements, exhibits the lowest execution time. RR, with four requirements, requires slightly more computational time, whereas the UAV benchmark, comprising twelve requirements, suffers a substantially higher cost. This is because each additional requirement introduces an additional predictive model whose confidence must be evaluated during the generation and refinement of candidate adaptations. As a result, a larger set of requirements increases the overall computational cost required to identify a satisfactory adaptation. Notice that RR (with four controllable features) exhibits a lower execution time than UAV (three controllable features). Although RR contains one additional controllable feature, it involves only four requirements compared with twelve requirements in UAV. As a result, the larger number of predictive models evaluated for UAV dominates the computational cost, showing execution time is determined jointly by the size of the search space and the number of requirements rather than by either one alone. The comparison between the UAV and the UAV2 provides additional evidence of scalability. Both benchmarks contain the same twelve requirements, differing only in the number of controllable features (three versus six). This experiment isolates the

[5] UAV2 has twice as many controllable features as UAV.




(a) Subject RR (b) Subject ADAS (c) Subject UAV

**Fig. 6.** Cost in terms of execution time (lower is better).




(a) Scalability in terms of controllable features (b) Scalability in terms of requirements

**Fig. 7.** Scalability analysis. Error bars represent interquartile ranges.

effect of the adaptation-space dimensionality. Results indicate that the computational cost of GLASS increases in a controlled manner as the complexity of the adaptation grows. It shows that the proposed GLASS remains computationally practical for runtime adaptation of all selected evaluation subjects.

> **RQ3 summary.** The results show that GLASS significantly reduces execution time compared to black-box optimization and achieves performance comparable to the fastest white-box baselines. This indicates that GLASS introduces a low planning overhead, making it a promising candidate for time-sensitive self-adaptation.

#### 5.2.4. Sensitivity analysis (RQ4)

To answer RQ4, we analyze how the configuration hyperparameters of GLASS affect the satisfaction likelihood and adaptation quality distance. We vary one parameter at a time while keeping all others at their default values. We evaluate $\tau \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$, $max_iterations \in \{2, 5, 10, 15, 20, 30\}$, $num_counterfactuals \in \{1, 3, 5, 8, 10\}$.

**Results.** We report the distributions of satisfaction likelihood and quality distance by varying the confidence threshold $\tau$, $max_iterations$, and $num_counterfactuals$ across the three subjects. Fig. 8 shows the distributions of satisfaction likelihood and quality distance obtained when varying the confidence threshold $\tau$. Varying $\tau$ between 0.5 and 0.9 yields identical distributions in all subjects. The median satisfaction likelihood remains approximately 0.60, 0.91, and 0.87 for ADAS, RR, and UAV, respectively, and the median quality distance remains approximately 0.50, 0.08, and 0.68.

Fig. 9 shows the sensitivity analysis of GLASS with respect to $max_iterations$. The search budget affects the results only when it is very small. With two iterations, for example, the counterfactual search often terminates before reaching a favorable region. In RR, the median quality distance increases from approximately 0.08 to 0.11, and the first quartile of the satisfaction likelihood decreases from approximately 0.78 to 0.76. In ADAS, the upper quartile of the quality distance increases from approximately 0.55 to 0.58. UAV is unaffected across the whole range.

Fig. 10 shows the sensitivity analysis of GLASS with respect to the factor $num_counterfactuals$. The number of counterfactuals has a negligible effect on satisfaction likelihood and a small effect on quality distance. In RR, the median satisfaction likelihood decreases only marginally (from approximately 0.92 to 0.90), while the median quality distance increases from approximately 0.06 with one counterfactual to approximately 0.10 with ten. In ADAS, the median quality distance increases from approximately 0.50 to 0.53, although a single counterfactual yields a wider spread (interquartile range approximately 0.35 to 0.55). UAV is unaffected.

> **RQ4 summary.** Increasing the number of counterfactuals does not substantially affect the satisfaction likelihood, although a slight increase in quality distance is observed for RR and ADAS. Overall, the results provide no evidence that subject-specific tuning is necessary for the considered subjects, suggesting that the same default configuration can be used across them.

### 5.3. Threats to validity

For practical reasons, all experiments are conducted in a simulated environment rather than on real-world systems. This choice constitutes a potential threat to external validity, as the results depend on a specific simulation platform. We mitigate this threat by evaluating GLASS on three publicly available, open-source CPS benchmarks that differ in the number of requirements and in the size of their controllable and observable feature spaces. The use of multiple, heterogeneous benchmarks improves the generalizability of the findings and enables replication by other researchers.

A potential threat to internal validity arises from the counterfactual generation process, which may emphasize changes to certain features while neglecting others. To mitigate this risk, we generate counterfactuals based on nearest-neighbor configurations, rather than relying on

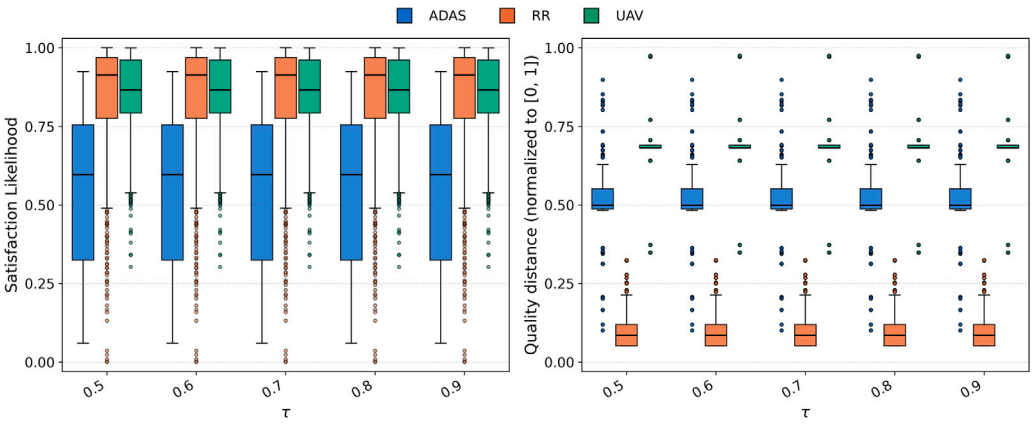



**Fig. 8.** Sensitivity analysis w.r.t. $\tau$.

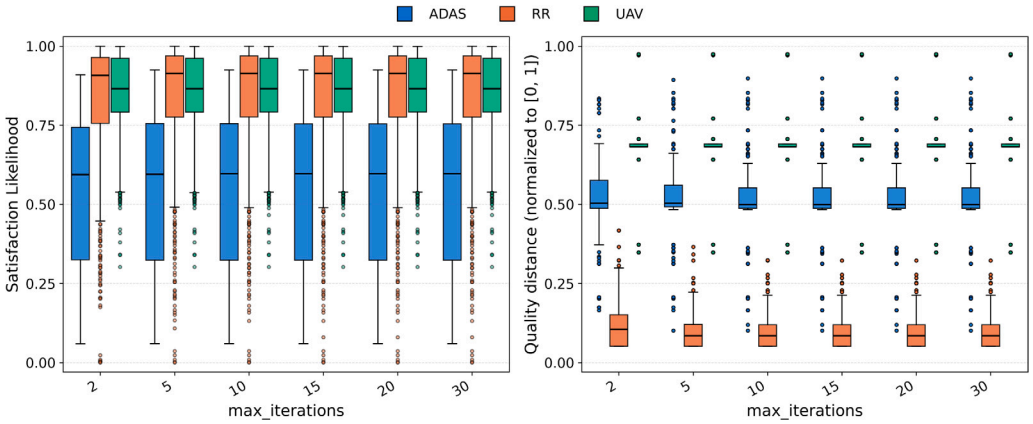



**Fig. 9.** Sensitivity analysis w.r.t. *max_iterations*.

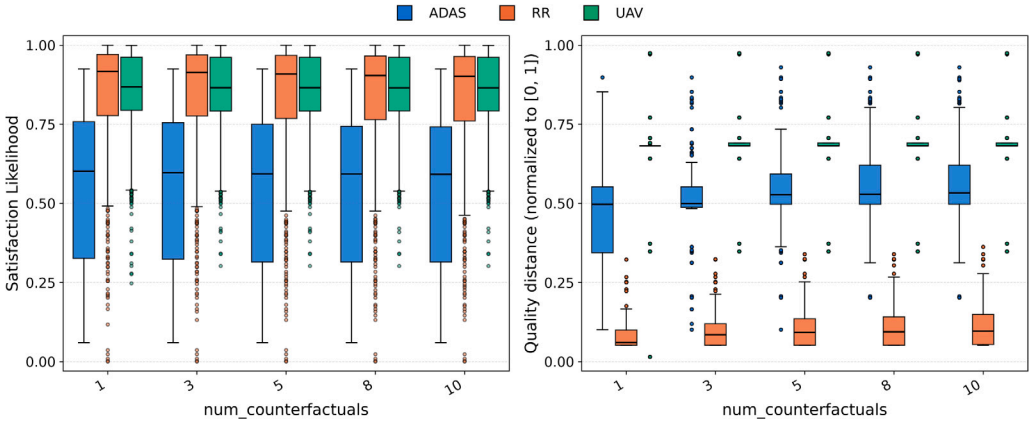



**Fig. 10.** Sensitivity analysis w.r.t. *num_counterfactuals*.

random or arbitrary feature changes. In addition, features are selected for modification according to their contribution to confidence loss, ensuring that adaptations are guided by features with a meaningful impact on requirement satisfaction.

We address construct validity threats by comparing GLASS against three baseline approaches using multiple, well-defined evaluation metrics. The simulated setting provides full control over both controllable and observable features, enabling consistent and repeatable experimentation. To enhance the reliability of our findings, each experiment is repeated 200 times. Statistical significance is assessed using the non-parametric Mann–Whitney U test.

## 6. Discussion

This section discusses GLASS and its underlying assumptions and limitations that should be considered when GLASS is deployed.

### 6.1. Assumptions and limitations

GLASS relies on assumptions common to learning-based self-adaptation. First, it assumes that the predictive models available in the Knowledge component remain representative of the operational environment after deployment. This assumption is reasonable when the

system behavior and its environment do not undergo substantial distribution shifts. Second, GLASS assumes that controllable feature domains are correctly specified, ensuring that all generated adaptations remain feasible. The approach also inherits the assumptions of the explanation techniques it employs. PDPs estimate marginal feature effects under a feature-independence assumption. Moreover, GLASS exploits PDPs only to guide the search toward promising regions rather than to replace the predictive models used to validate candidate adaptations.

### 6.2. Generalizability of our results

Our empirical evaluation is currently limited to simulation-based cyber–physical system (CPS) benchmarks. Although these benchmarks span different levels of complexity, GLASS is not inherently specific to CPSs and may generalize to other classes of software-intensive autonomous systems, provided that the assumptions underlying the approach hold (e.g., the availability of predictive models, well-defined controllable features, and runtime adaptation through a MAPE-K feedback loop). Nevertheless, such generality has not yet been experimentally validated. Future work will therefore investigate the applicability of GLASS to additional application domains, including real-world deployments and alternative autonomous software systems, to further assess its robustness and external validity.

## 7. Related work

This section provides a brief overview of existing work on foundational perspectives of explainability and its integration into autonomous and adaptive systems. We have categorized it into different sections, as follows.

### 7.1. Explainable AI

Explainable AI refers to the ability of an AI model to be transparent and understood by different stakeholders involved in it [29]. This field has emerged as crucial for making black-box predictive models interpretable. Indeed, predictive models having an opaque nature (e.g., deep neural networks) reduce user trust and also hinder the ability to make effective decisions [30].

The usage of AI components in software systems is increasing; therefore, explainability is now gaining attention in the field of software engineering as well [31,32]. Foundational research in explainable AI [33,34] has introduced a variety of techniques for interpreting machine learning models, including feature attribution methods, example-based explanations, counterfactual explanations, and global explanation techniques. These studies primarily focus on improving the interpretability and transparency of machine learning models and establishing methods for evaluating explanation quality. GLASS builds upon existing model-agnostic explanation techniques by integrating them into the feedback loop to support runtime adaptation decisions.

### 7.2. Explainability in self-adaptive systems

Self-Adaptive systems need to adjust themselves in response to changes in their surroundings, operating conditions, or user requirements. Most often, these systems utilize black-box predictive models to make complex decisions. For this reason, the stakeholders (system engineers and end users) may find difficulty in understanding why such a system is adapted in a certain manner, especially in safety-critical domains where each decision has a considerable impact [35]. Thus, integrating the notion of explainability into adaptive systems has gained momentum recently.

Diallo et al. [36] propose a mechanism that utilizes explainable AI to build user trust in a deep learning model, as well as to reduce the search space of adaptations. Blumreiter et al. [37] presented the MAB-EX framework that aims to make cyber–physical systems explainable at runtime. MAB-EX aims to answer the questions and explain the system behavior to the stakeholders. The author in [38] argues that explanations come with associated costs, and also, there is a possibility that humans may make bad judgments. Therefore, they believe that it is not always necessary that explanations be helpful, and also, which explanation is to be provided to the operator. To address this concern, they propose a formal framework to reason about the behavior of an adaptive system and to characterize explanations with regard to content and cost. Wohlrab et al. [39] propose an approach that makes use of machine learning techniques to explain and reason about automated planning. They identified the trade-offs between quality attributes to assist humans in understanding the adaptation behavior. XRL-DINE [40] is proposed, which combines explainable reinforcement learning techniques to make adaptation decisions transparent, thus enhancing user trust as well as assisting in the debugging of self-adaptive systems. The XSA [7] framework is proposed, which is tailored to achieve different levels of explainability and provides proof of concept to give an explanation to adaptation decisions in the domain of robotics. There could be different human roles regarding the adaptation loop, and XSA differentiates between these different roles, such as "out-of-the-loop" users (who do not involve themselves in the adaptation process) and "in-the-loop" users (who may monitor or supervise the adaptation process) [41,42]. Based on that, EASE [43] — a framework that combines statistical model checking and interpretable machine learning — is proposed, which enforces dependability via formal analysis to enhance the level of trust in human–machine teaming. Negri et al. [44] presented the XDA approach that incorporates an interpretable machine learning technique into the feedback control loop. It not only makes the black-box model transparent, but it also reduces the computational cost by reducing the search space of controllable features. The authors validated their approach in the robotics domain.

Existing explanation-driven adaptation approaches leverage interpretability to improve runtime adaptation; they essentially rely on either local or global explanatory information. XDA employs global model explanations (PDPs) to guide white-box optimization but lacks instance-level guidance; CSA-$\Phi$ leverages counterfactual explanations for minimal, instance-specific adaptations but provides no global perspective. In contrast, GLASS combines both explanation types as machine-readable artifacts within the runtime optimization pipeline. Moreover, GLASS recognizes computational efficiency as a first-class concern, critical for CPS, by separating expensive PDP computation (in the offline phase) from the online adaptation.

## 8. Conclusion

We introduce GLASS, a novel two-stage, explanation-driven self-adaptation approach that improves both the transparency and efficiency of runtime adaptation decisions. GLASS integrates joint global–local explanations into the feedback control loop to guide adaptation at runtime. Local explanations in the form of counterfactuals are exploited to craft adaptation decisions, while global explanations derived from PDPs provide a broader understanding of the model's behavior to support informed adaptation when local information is insufficient.

We evaluate GLASS through an empirical study using three evaluation subjects (open-source CPS benchmarks) and compared it against three baseline methods, including XDA, CSA-$\Phi$, and a black-box optimization method (NSGA-III). The results show that GLASS achieves significantly lower or comparable execution times relative to the baselines. In terms of effectiveness, GLASS consistently delivers competitive or superior performance with respect to satisfaction likelihood and success rate, while maintaining robust quality distance across varying levels of subject complexity.

We plan to extend GLASS by incorporating other optimization strategies, such as Bayesian optimization, which could be employed to further refine search within the explanation-guided feature space. Finally, integrating additional model-agnostic interpretability techniques — such as SHAP values and feature importance measures — represents a promising direction for providing richer and more heterogeneous explanations to support transparent self-adaptation.

## CRediT authorship contribution statement

**Ehsan Elahi:** Writing – original draft, Validation, Software. **Matteo Camilli:** Writing – review & editing, Validation, Supervision, Methodology, Conceptualization. **Raffaela Mirandola:** Writing – review & editing, Supervision, Methodology, Funding acquisition.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work was partially supported by the Helmholtz Association (HGF) with the KiKIT project and the Grant 46.23 (Engineering Secure Systems) and by the German Research Foundation (DFG) - SFB 1608 - 501798263.

## Data availability

Link to replication package (code, data) provided

Replication package (Original data) (Zenodo)

## References

[1] S.M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, D. Perez-Palacin, Uncertainty in self-adaptive systems: A research community perspective, ACM Trans. Auton. Adapt. Syst. (TAAS) 15 (4) (2021) 1–36.

[2] D. Weyns, R. Calinescu, R. Mirandola, K. Tei, M. Acosta, N. Bencomo, A. Bennaceur, N. Boltz, T. Bures, J. Camara, et al., Towards a research agenda for understanding and managing uncertainty in self-adaptive systems, ACM SIGSOFT Softw. Eng. Notes 48 (4) (2023) 20–36.

[3] F.R. Negri, N. Nicolosi, M. Camilli, R. Mirandola, 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, Association for Computing Machinery, New York, NY, USA, 2024, pp. 189–199, http://dx.doi.org/10.1145/3643915.3644085.

[4] M.A. Langford, B.H. Cheng, "Know what you know": Predicting behavior for learning-enabled systems when facing uncertainty, in: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, IEEE, 2021, pp. 78–89.

[5] O. Gheibi, D. Weyns, Lifelong self-adaptation: Self-adaptation meets lifelong machine learning, in: Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2022, pp. 1–12.

[6] F. Cantucci, M. Marini, R. Falcone, Trustworthiness assessment of an adaptive and explainable robot in a real environment, Int. J. Soc. Robot. (2025) 1–12.

[7] M. Camilli, R. Mirandola, P. Scandurra, XSA: Explainable self-adaptation, in: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022, pp. 1–5.

[8] R. Straub, F. Sihler, A. Torbati, C. Wang, R. Groner, V. Klös, M. Tichy, Explainability in self-adaptive systems: A systematic literature review, in: D. Taibi, D. Smite (Eds.), Software Engineering and Advanced Applications, Springer Nature Switzerland, Cham, 2026, pp. 280–297.

[9] C. Molnar, Interpretable Machine Learning, Lulu. com, 2020.

[10] F. Negri, N. Nicolosi, M. Camilli, R. Mirandola, Efficient self-adaptation through explanation-driven white-box optimization, ACM Trans. Auton. Adapt. Syst. (2025) http://dx.doi.org/10.1145/3737648, Just Accepted.

[11] E. Elahi, M. Camilli, R. Mirandola, Counterfactual self-adaptation in cyber-physical systems, in: D. Taibi, D. Smite (Eds.), Software Engineering and Advanced Applications, Springer Nature Switzerland, Cham, 2026, pp. 219–236.

[12] S. Wachter, B. Mittelstadt, C. Russell, Counterfactual explanations without opening the black box: Automated decisions and the GDPR, Harv. JL Tech. 31 (2017) 841.

[13] N. Esfahani, S. Malek, Uncertainty in self-adaptive software systems, in: Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers, Springer, 2013, pp. 214–238.

[14] O. Gheibi, D. Weyns, F. Quin, Applying machine learning in self-adaptive systems: A systematic literature review, ACM Trans. Auton. Adapt. Syst. (TAAS) 15 (3) (2021) 1–37.

[15] H. Jilderda, C. Raibulet, Mape-k based guidelines for designing reactive and proactive self-adaptive systems, in: European Conference on Software Architecture, Springer, 2023, pp. 53–68.

[16] T.M. Mitchell, Machine Learning, first ed., McGraw-Hill, Inc., USA, 1997.

[17] E. Elahi, S. Kanwal, A.N. Asif, A new ensemble approach for software fault prediction, in: 2020 17th International Bhurban Conference on Applied Sciences and Technology, IBCAST, IEEE, 2020, pp. 407–412.

[18] C. Wang, A. Torbati, V. Klös, Exploring explainability requirements for self-adaptive systems, in: Euromicro Conference on Software Engineering and Advanced Applications, Springer, 2025, pp. 237–253.

[19] T. Miller, Explanation in artificial intelligence: Insights from the social sciences, Artificial Intelligence 267 (2019) 1–38.

[20] F. Doshi-Velez, B. Kim, Towards a rigorous science of interpretable machine learning, 2017, arXiv preprint arXiv:1702.08608.

[21] K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints, IEEE Trans. Evol. Comput. 18 (4) (2013) 577–601.

[22] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, second ed., Springer Publishing Company, Incorporated, 2015.

[23] S. Nejati, L. Sorokin, D. Safin, F. Formica, M.M. Mahboob, C. Menghi, Reflections on surrogate-assisted search-based testing: A taxonomy and two replication studies based on industrial adas and simulink models, Inf. Softw. Technol. 163 (2023) 107286.

[24] M. Camilli, R. Mirandola, P. Scandurra, Enforcing resilience in cyber-physical systems via equilibrium verification at runtime, ACM Trans. Auton. Adapt. Syst. 18 (3) (2023) 1–32.

[25] G. Moreno, C. Kinneer, A. Pandey, D. Garlan, DARTSim: An exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, IEEE, 2019, pp. 181–187.

[26] J. Blank, K. Deb, Pymoo: Multi-objective optimization in python, IEEE Access 8 (2020) 89497–89509.

[27] A. Arcuri, L. Briand, 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, Association for Computing Machinery, New York, NY, USA, 2011, pp. 1–10, http://dx.doi.org/10.1145/1985793.1985795.

[28] H.B. Mann, D.R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, Ann. Math. Stat. 18 (1) (1947) 50–60, (Accessed 2023-07-20).

[29] M.T. Ribeiro, S. Singh, C. Guestrin, " why should i trust you?" explaining the predictions of any classifier, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135–1144.

[30] P.P. Angelov, E.A. Soares, R. Jiang, N.I. Arnold, P.M. Atkinson, Explainable artificial intelligence: an analytical review, Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 11 (5) (2021) e1424.

[31] C.K. Tantithamthavorn, J. Jiarpakdee, Explainable AI for software engineering, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE, 2021, pp. 1–2, http://dx.doi.org/10.1109/ASE51524.2021.9678580.

[32] S. Speth, S. Stieß, S. Becker, A vision for explainability of coordinated and conflicting adaptions in self-adaptive systems, in: Proceedings of 14th Central European Workshop on Services and their Composition, ZEUS 2022, 2022, pp. 16–19.

[33] L. Longo, M. Brcic, F. Cabitza, J. Choi, R. Confalonieri, J.D. Ser, R. Guidotti, Y. Hayashi, F. Herrera, A. Holzinger, R. Jiang, H. Khosravi, F. Lecue, G. Malgieri, A. Páez, W. Samek, J. Schneider, T. Speith, S. Stumpf, Explainable artificial intelligence (XAI) 2.0: A manifesto of open challenges and interdisciplinary research directions, Inf. Fusion 106 (2024) 102301, http://dx.doi.org/10.1016/j.inffus.2024.102301, URL https://www.sciencedirect.com/science/article/pii/S1566253524000794.

[34] M. Langer, D. Oster, T. Speith, H. Hermanns, L. Kästner, E. Schmidt, A. Sesing, K. Baum, 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, http://dx.doi.org/10.1016/j.artint.2021.103473, URL https://www.sciencedirect.com/science/article/pii/S0004370221000242.

[35] M. Blumreiter, J. Greenyer, F.J.C. Garcia, V. Klös, M. Schwammberger, C. Sommer, A. Vogelsang, A. Wortmann, Towards self-explainable cyber-physical systems, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C, IEEE, 2019, pp. 543–548.

[36] A.B. Diallo, H. Nakagawa, T. Tsuchiya, Adaptation space reduction using an explainable framework, in: 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC, 2021, pp. 1653–1660, http://dx.doi.org/10.1109/COMPSAC51774.2021.00247.

[37] M. Blumreiter, J. Greenyer, F.J. Chiyah Garcia, V. Klös, M. Schwammberger, C. Sommer, A. Vogelsang, A. Wortmann, Towards self-explainable cyber-physical systems, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C, 2019, pp. 543–548, http://dx.doi.org/10.1109/MODELS-C.2019.00084.

[38] N. Li, S. Adepu, E. Kang, D. Garlan, 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, 2020, pp. 181–187.

[39] R. Wohlrab, J. Cámara, D. Garlan, B. Schmerl, Explaining quality attribute tradeoffs in automated planning for self-adaptive systems, J. Syst. Softw. 198 (2023) 111538.

[40] F. Feit, A. Metzger, K. Pohl, Explaining online reinforcement learning decisions of self-adaptive systems, in: 2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS, IEEE, 2022, pp. 51–60.

[41] N. Li, J. Cámara, D. Garlan, B. Schmerl, Z. Jin, Hey! preparing humans to do tasks in self-adaptive systems, in: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, IEEE, 2021, pp. 48–58.

[42] R. De Lemos, Human in the loop: What is the point of no return? in: Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, 2020, pp. 165–166.

[43] M.M. Bersani, M. Camilli, L. Lestingi, R. Mirandola, M. Rossi, P. Scandurra, Towards better trust in human-machine teaming through explainable dependability, in: 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), 2023, pp. 86–90, http://dx.doi.org/10.1109/ICSA-C57050.2023.00029.

[44] F.R. Negri, N. Nicolosi, M. Camilli, R. Mirandola, 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, 2024, pp. 189–199.