Counterfactual Self-adaptation in Cyber-Physical Systems

Ehsan Elahi[1][0009−0008−7931−8404], Matteo Camilli[2][0000−0003−2491−5267], and Raffaela Mirandola[1][0000−0003−3154−2438]

[1] Karlsruhe Institute of Technology, Karlsruhe, Germany
ehsan.elahi@kit.edu, raffaela.mirandola@kit.edu
[2] Politecnico di Milano, Milan, Italy
matteo.camilli@polimi.it

Abstract. Cyber-physical systems (CPS) operate in dynamic and uncertain environments, where maintaining operational objectives without manual intervention is critical. Self-adaptive systems (SAS) have emerged as a promising solution, leveraging machine learning (ML) models within feedback control loops to make runtime adaptation decisions. However, the black-box nature of these models poses challenges related to transparency and efficiency, particularly in safety-critical domains.
This paper introduces CSA-Φ, a counterfactual-based self-adaptation approach that integrates model-agnostic interpretable ML into the adaptation loop. CSA-Φ includes two main components: an offline pre-processor with pre-trained classifiers for requirement evaluation, and an online opportunistic adaptation engine that uses counterfactual explanations to guide adaptation decisions efficiently. By bridging the gap between explainability and actionable adaptation, CSA-Φ enhances decision-making while reducing adaptation cost. Experimental results show that CSA-Φ achieves significant improvements in execution efficiency and provides adaptation decisions that are both effective and interpretable, outperforming selected baseline approaches.

Keywords: Explainability · counterfactual explanations · cyber-physical systems · self-adaptive systems

1 Introduction

Cyber-physical systems (CPS) are software-intensive systems that tightly integrate computational processes with physical components. These systems typically operate in dynamic environments where uncertainty arises from changing or unpredictable conditions. As CPS and software systems become increasingly prevalent in critical domains, ensuring they can continuously maintain their operational objectives without requiring manual intervention or system redeployment is essential.

To address these challenges, self-adaptive systems (SAS) [10] have emerged as a promising solution. These systems are designed to autonomously adjust their behavior in response to environmental changes or internal anomalies, making them well-suited for managing uncertainty in real-time [29]. Central to the architecture of

SAS is an adaptation layer, which monitors system behavior and triggers adjustments when necessary through a feedback control loop. This loop operates by observing the current state, analyzing deviations, planning corrective actions, and executing adaptations—often leveraging machine learning (ML) techniques to handle complex decision-making tasks [15, 21, 31].

Supervised learning methods such as neural networks [27] and ensemble models [14] are commonly employed to build predictive models within the feedback loop encapsulating decision logic for adaptation. While ML-based adaptation offers benefits like learning from past behavior, it also poses transparency challenges. ML models are often black boxes, making it difficult for system engineers or administrators to understand adaptation decisions. This lack of interpretability can undermine trust in the system and lead to risks in safety-critical domains. In response to these concerns, the concept of explainability has gained traction in the context of self-adaptive systems [24]. Explainability aims to make the decision-making process more transparent and understandable to human stakeholders. Recent works [7, 18, 32] demonstrate how integrating explainability into the adaptation loop can enhance user trust, support debugging, and guide more informed adaptation strategies.

Beyond transparency, efficiency is another critical consideration when using black-box models for runtime adaptation. These models often need to explore large, complex solution spaces involving multiple, sometimes conflicting objectives (e.g., performance, reliability, energy efficiency). Exhaustive search in such spaces is computationally infeasible, highlighting the need for efficient optimization strategies. Explainability can also contribute here by revealing meaningful relationships between system parameters and objectives, thereby reducing the search space, guiding adaptation, and accelerating convergence toward high-quality solutions [7, 15].

In this paper, we explore the use of counterfactual explanations [28]—a model-agnostic, interpretable machine learning technique—in the context of self-adaptive systems. Being model-agnostic, counterfactual explanations can be consistently applied to any machine learning model without assuming anything about the specific underlying model. Building on prior work [25] that employs post hoc explanation methods such as Partial Dependence Plots (PDP) [13] to enhance system interpretability, we introduce CSA-Φ[1], a novel counterfactual-based self-adaptation approach. CSA-Φ aims to bridge the gap between explainability and actionable adaptation strategies by leveraging counterfactual reasoning to support informed, interpretable, and effective adaptation decisions. The core objective is to demonstrate how counterfactual explanations can enhance the transparency of adaptive behavior while ensuring that multiple system requirements are satisfied efficiently and reliably.

The main components of CSA-Φ are *i*) an *offline pre-processor*, which includes a set of pre-trained classifiers, each specifically tailored to evaluate a distinct system requirement. These classifiers label the satisfaction of each requirement as either *true* (satisfied) or *false* (violated); and *ii*) an *online opportunistic adaptation engine*, which operates at runtime to generate and utilize explanations in order to guide and refine the selection among potential adaptation options.

The main contributions of this work are given as follows:

[1] CSA-Φ stands for Counterfactual Self-Adaptation in cyber-PHYsical systems

- We propose CSA-Φ, an automated approach that integrates model-agnostic interpretable machine learning into the feedback control loop to steer adaptation decisions while satisfying multiple system requirements.
- We conduct extensive experiments to evaluate the performance of CSA-Φ, demonstrating its efficiency and effectiveness in comparison to selected baseline approaches.
- We illustrate the usefulness of counterfactual explanations in enhancing the quality of adaptation decisions, by providing actionable insights into why certain adaptations are preferable.

The rest of this paper is organized as follows: Section 2 describes the illustrative example leveraged in our work. In section 3, some background knowledge is introduced to help the readers. The proposed approach CSA-Φ is illustrated in section 4, while empirical evaluation is discussed in section 5 of this paper. In section 6 threats to validity are discussed whereas section 7 throws some light on the recent literature. Section 8 has the conclusion of the work.

2 Illustrative example

As a running example, we consider the search-and-rescue robotic system, which is widely used in the literature [12]. In this example, there is a robotic system to discover and rescue people in dangerous situations such as fire, earthquake, hurricane, etc. This robotic system consists of appropriate abstractions for the physical interfaces such as sensors and actuators, as well as it incorporates the essential functions of rescuing such as navigation, human and/or obstacle detection, and avoidance. Since the robotic system has to operate in uncertain situations with different changes in the environment and resource variability at runtime, therefore, its components are designed to be configured at runtime. It is an adaptation layer, which is equipped with the feedback control loop, that enacts these configuration changes to meet the adaptation goals. Examples of the adaptation goals could be the safety requirements such as *"the probability that rescue robot maintains the protective distance from humans while moving must be greater than 0.90"* (R2) or it could be *"the probability that rescue robot does not crash into an obstacle must be greater than 0.90"* (R4). Inside the control loop, there are pre-trained classifiers (each trained for different requirements) that predict the requirements not satisfied based on the operating conditions and then, accordingly, steer the changes in the configuration so that the probability of satisfying the requirements can be increased, as per the operating conditions.

The operating conditions consist of configuration dimensions and environment dimensions. In the configuration dimension, there are features that the system can change to increase the likelihood of satisfying requirements, therefore also known as controllable features (CFs). On the other hand, environment dimensions are those that can only be observed as they are related to the environment, therefore also known as observable features (OFs). In Table 1, CFs (i.e., cruise speed, image resolution, illuminance, and controls responsiveness) and OFs (i.e., power, smoke intensity, obstacle size, obstacle distance, firm obstacle) are presented, which collectively form the semantic space [8].

Table 1: Semantic space of the self-adaptive system

Feature	Dimension	Type	Range
cruise speed	CF	continuous	[0, 5] m/s
image resolution	CF	categorical	{low, mid, high}
control responsiveness	CF	continuous	[10, 50] Mbit/s
illuminance	CF	continuous	[40, 120000] lux
power	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

3 Background

In this section, we introduce background concepts to make the paper self-contained. Specifically, we describe the concepts of a self-adaptive system, supervised and interpretable machine learning.

3.1 Self-Adaptive System

Self-adaptive systems adjust their behavior in response to changes in the environment in which they are deployed or in the system itself. The conceptual model of a self-adaptive system [29] has four elements: environment, managed system, adaptation goals, and feedback loop.

The environment is the external world, including both virtual and physical entities and users, where the system operates. It is not controlled by the system and can only be sensed or influenced through sensors and effectors, leading to some uncertainty. The managed system is the application software that serves users and can be adapted via its sensors and effectors. It is controllable and subject to adaptation. Adaptation goals are the managing system's quality concerns for the managed system, represented as uncertainties to address during operation. The managing system consists of adaptation goals and a feedback loop (MAPE-K: Monitor, Analyze, Plan, Execute, Knowledge) [17], which monitors and adapts the managed system and its environment to achieve these goals.

3.2 Supervised Machine Learning

In the context of machine learning, classification is a predictive task of anticipating the class or label y of the given data point or instance x, having multiple feature values. Supervised learning techniques are used for the classification task for predicting either binary or multiclass labels. In other words, we have an instance x which consists of different features having some specific value x_j for each given feature j. A classifier is a supervised learning technique that performs the classification task of predicting the class labels.

In supervised learning, there is training data having class labels for each instance; it is the training set on which the classifier is trained (learning from the data). Once the classifier is trained, it predicts the testing data, which it has not seen earlier. Hence, it

is evaluated on how accurately it can predict the unseen data. In supervised machine learning, there are many different classifiers for different problems. Some of them are simple models having predictions easy to understand and to explain, such as linear regression. On the other hand, there are some complex predictive models that are not inherently explained and therefore known as black box models, e.g., neural networks.

3.3 Interpretable Machine Learning

Interpretable simply refers to the extent of understanding of some phenomenon by humans. Similarly, Interpretable ML [24] is the process of extracting useful insights from the machine learning model, i.e., predictions, so that they can be well understood by humans as to why the particular decision has been made [2]. According to Miller [23], we make use of the terms *explainable* and *interpretable* interchangeably in this work. Interpretability can be global or local. Global interpretability gives a holistic view of the model and, therefore, explains the average behavior of the model. On the other hand, local interpretability explains the single prediction.

Counterfactual explanations, which come under the umbrella of local interpretability, demonstrate the causal situation by giving insights into the occurrence of the prediction. For example, "if X had not happened, Y would not have happened". Here, X is the input instance that causes the Y output. To describe it in mathematical form:

$$f(X) = Y \quad \text{(Original Output)}$$
$$f(X') = Y' \quad \text{(Desired Output)}$$

where X' is obtained by slightly (minimally) modifying X.

If, for example, the robot is operating in darkness, then it may need to increase its camera resolution and illuminance to better detect and classify the obstacle and/or humans. The possible scenario of counterfactual explanation could be to slightly change the controllable feature values, such as *image resolution* and *illuminance*, so that the output of satisfying the system requirement is flipped from *False* to *True*. It is important to note that the counterfactual should be as close as possible to the original values and change as minimum features as possible for the given instance.

Figure 1 describes visually how counterfactual explanations work for the given model's decision boundary. For example, in the Fig. 1, the original input actually falls into the class label, *False*, the insights gained from counterfactuals guide the change in the value of the input feature *cruise speed*, so that the system requirement is satisfied by having the class label *True* (desired class).

4 Proposed Approach

This section presents our CSA-Φ approach that embeds counterfactual explanations into the MAPE-K loop, thus transforming insights into actionable adaptation configurations. It enables the system to not only reason about why an adaptation is necessary but also to answer what changes are needed to have the adaptation that satisfies the requirements. The knowledge component of the MAPE-K loop integrates the set of

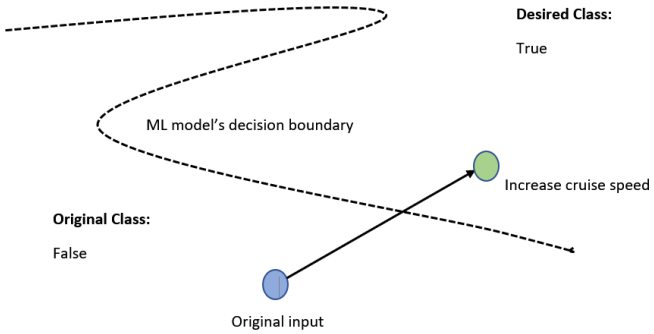

Fig. 1: Illustration of counterfactual explanation in a machine learning model.

predictive models. As CSA-Φ relies on a model-agnostic, interpretable ML technique, it adds flexibility in using any underlying predictive model consistently and smoothly. Moreover, we have trained separate predictive models for each requirement to support multi-requirements scenarios, as well as to handle the trade-offs among requirements. This enables requirements-aware explainable systems, e.g., safety-critical systems in which adaptation may have competing priorities.

CSA-Φ has two components: an offline preprocessor and an online opportunistic approach. These components work in two different phases of the system. Offline pre-processor works before the system is deployed in its working environment. The online opportunistic approach is executed along with the MAPE-K control loop when the system is deployed in its running environment. Figure 2 shows the high-level view of the CSA-Φ approach and how it is incorporated into the MAPE-K control loop. An in-depth discussion of these aspects is given in the following sections.

4.1 Offline pre-processor

This component is designed to preprocess the imbalanced data and train the models offline to enable the self-adaptive system to be computationally efficient and explainable. Algorithm 1 describes the overall working of the pre-processor component. Given as input controllable features CFs, system requirements $reqs$, and the balance ratio $balance\_ratio$, the offline pre-processor outputs the resampled data, as well as a set of trained models, to be utilized in the online phase. Algorithm 1 has four steps, as follows.

Step 1: The first step involves finding similar (nearby) data instances for a given test input. To achieve this, we employ the k-nearest neighbors (KNN) algorithm [9], which is used to identify instances that are close in the feature space. Using KNN ensures that the generated counterfactuals are both feasible and realistic, as they are based on actual data points rather than arbitrary perturbations. This step is implemented in Lines 2–4 of Algorithm 1.

Step 2: In this step, we compute the class distribution for each requirement by counting the number of instances labeled as true and false. The minority class is identified by comparing these counts. Based on the imbalance, a sampling strategy is defined using a balance ratio, which determines the extent to which the minority

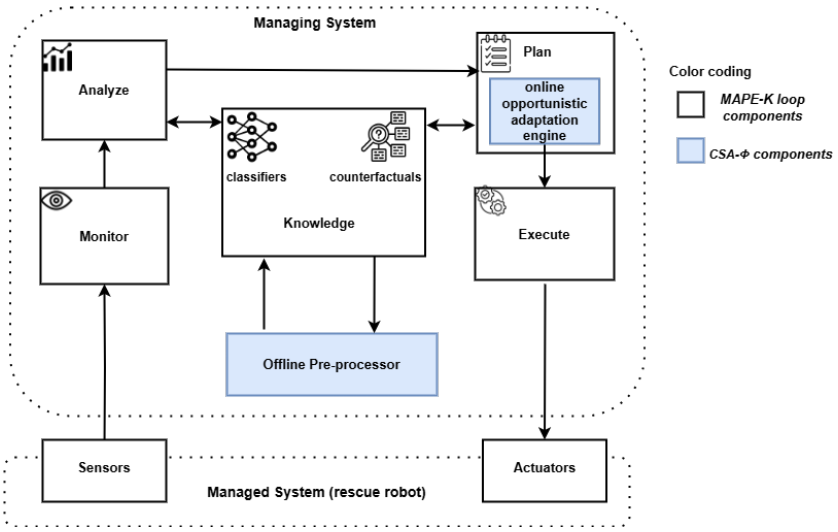

Fig. 2: CSA-Φ: Integrating explainability into the self-adaptive system.

class should be oversampled. This step enables us to control the number of synthetic samples generated for the minority class. Refer to Lines 5–9 of Algorithm 1.

Step 3: To address class imbalance, we apply Borderline-SMOTE [16], a resampling technique that focuses on generating synthetic instances near the decision boundary. It interpolates new samples between borderline minority instances and their nearest neighbors in the feature space. Since each requirement may have a different degree of class imbalance, a customized sampling strategy is applied for each one. Refer to Lines 10–15 of Algorithm 1.

Step 4: In the final step, a separate neural network model is trained for each requirement using the balanced dataset. For instance, if we have four distinct requirements, four corresponding models are trained. These models are later used in the online phase of the system, allowing for efficient reuse and reduced computational overhead during runtime adaptation. See Lines 16–23 of Algorithm 1 for implementation details.

4.2 Online opportunistic engine

This component is responsible for determining the optimal values of controllable features, thereby guiding the adaptive behavior of the system to ensure that all specified requirements are satisfied. The objective is to identify an effective adaptation strategy that brings the system back into a compliant state. The overall functionality of the online opportunistic component is described in Algorithm 2. We assume that this component is integrated into the Plan phase of the MAPE-K control loop. Accordingly, when the predictive model outputs a negative label—indicating that one or more requirements are currently violated—the Analyze phase of the loop activates the Plan phase. At this point, the online opportunistic engine is invoked to compute a suitable adaptation.

Algorithm 1 Offline Pre-processing

Input: Input features X; Outcome labels y; Controllable features $CF = \{CF_1, \ldots, CF_m\}$; System requirements $reqs$; Balance ratio $balance\_ratio$.
Output: Trained models $models = \{M_1, \ldots, M_n\}$; Resampled datasets ($X\_resampled\_list$,$y\_resampled\_list$).

1: Initialize empty lists: $X\_resampled\_list \leftarrow \emptyset$, $y\_resampled\_list \leftarrow \emptyset$
2: **Step 1: Train KNN Model**
3: $knn \leftarrow$ KNearestNeighbor()
4: knn.fit(X,num_neighbors = 0)
5: **Step 2: Class Distribution Analysis**
6: **for** each requirement req in $reqs$ **do**
7: $y_{req} \leftarrow$ extract_labels(y,req)
8: $minority\_class \leftarrow$ identify_minority_class(y_{req})
9: $sampling\_strategy \leftarrow$ calculate_sampling_strategy($minority\_class$)
10: **Step 3: Data Resampling**
11: $sampler \leftarrow$ BorderlineSMOTE($sampling\_strategy = sampling\_strategy$)
12: ($X_{resampled}$,$y_{resampled}$) $\leftarrow$ sampler.fit_resample(X,y_{req})
13: $X\_resampled\_list$.append($X_{resampled}$)
14: $y\_resampled\_list$.append($y_{resampled}$)
15: **end for**
16: **Step 4: Model Training**
17: Initialize $models \leftarrow \emptyset$
18: **for** each req in $reqs$ **do**
19: $model \leftarrow$ NeuralNetwork()
20: model.train($X\_resampled\_list$[req],$y\_resampled\_list$[req])
21: $models$.append($model$)
22: **end for**
23: **Return** ($models$,$X\_resampled\_list$,$y\_resampled\_list$)

Algorithm 2 consists of the following three main steps:

Step 1: This step begins with exploring nearby instances in the feature space to generate a set of potential adaptations. Initially, the current system state serves as the baseline adaptation, which is incrementally refined as the algorithm progresses. To identify similar instances, the KNN algorithm is employed. For each neighbor identified, counterfactual examples are generated—these provide guidance on how to adjust the controllable feature values in order to satisfy system requirements. Each counterfactual corresponds to a potential adaptation, and these are collected into a candidate pool. This process is outlined in Lines 2–11 of Algorithm 2.

Step 2: In this step, a subset of controllable features is selected for modification based on the goal of minimizing confidence loss for the given requirement. Features that contribute least to the confidence degradation are prioritized for modification. These selected features are then updated using values suggested by the counterfactuals. After applying the changes, the model predicts the new confidence level for the requirement. If the updated state still fails to satisfy the requirement, the adaptation is rolled back to maintain system stability. See Lines 12–27 of Algorithm 2.

Step 3: Finally, the best adaptation, denoted as S_{opt}, is selected from the pool of potential adaptations. The selection criterion is based on the highest predicted confidence with respect to the system requirements. This ensures that the chosen adaptation not only satisfies the requirements but also does so with the greatest level of assurance. The final adaptation leverages insights from counterfactual explanations to adjust the system's controllable features. This step corresponds to Lines 28–30 of Algorithm 2.

Algorithm 2 Online Opportunistic Approach

Input: Current system state S; Set of controllable features CF; Trained models $models = \{M_1,...,M_n\}$; Set of nearest neighbors N.
Output: Opportunistic adaptation S_{opt}.

Initialize $S_{opt} \leftarrow S$
Step 1: Select Neighboring Adaptations
$neighbors \leftarrow$ Find nearest instances from N
$adaptations \leftarrow \{S\}$
for $neighbor \in neighbors$ **do**
    Generate counterfactuals $counterfactuals \leftarrow counterfactual[neighbor]$
    for $f \in counterfactuals$ **do**
        Generate candidate adaptation S'
        $adaptations \leftarrow adaptations \cup \{S'\}$
    end for
end for
Step 2: Opportunistic Loop:
for $S' \in adaptations$ **do**
    $excludedFeatures \leftarrow \emptyset$, $tempExcludedFeatures \leftarrow \emptyset$
    while number of excluded features $<$ total controllable features **do**
        Select features to modify based on minimizing confidence loss
        Modify feature value within bounds
        Update confidence C'
        if C' does not satisfy requirements **then**
            Revert to the previous adaptation
            $tempExcludedFeatures \leftarrow tempExcludedFeatures \cup \{feature\}$
        else
            $tempExcludedFeatures \leftarrow \emptyset$
        end if
    end while
    Compute confidence $\mathcal{F}(S')$
end for
Step 3: Select Best Adaptation
$S_{opt} \leftarrow$ adaptation with highest $\mathcal{F}(S')$
Return S_{opt}

5 Evaluation

In this section, an empirical evaluation of our CSA-Φ approach is presented, considering multiple requirements. Table 2 presents the requirements that are considered in this study. In our experiments, the simulation environment allows us to monitor all variables in the semantic space (Table 1) to determine the unsatisfied requirements in the simulated scenario. The experimental results answer the following research questions.

RQ1: What is the computational cost of CSA-Φ with respect to the selected baselines?

RQ2: How effective is the counterfactual-guided self-adaptive system compared to the baselines?

We conduct a series of experiments to evaluate the effectiveness and efficiency of CSA-Φ, comparing it against two baseline methods: *i*) XDA [25], an explanation-driven approach designed to guide adaptation decisions based on interpretability insights; and *ii*) NSGA-III [11], a multi-objective metaheuristic optimization technique that relies on predictions from black-box models. To address RQ1, we measure the execution time (in seconds) for CSA-Φ and both baselines to assess their computational efficiency. To address RQ2, we evaluate the satisfaction likelihood, defined as the probability that a specific requirement is fulfilled under the current operating conditions, as estimated by

Table 2: Description of requirements

#	Requirements
R1	The rescue robot must correctly classify obstacles with a probability greater than 0.85.
R2	The rescue robot must maintain a safe distance from humans while in motion with a probability greater than 0.90.
R3	The rescue robot must avoid unwanted contact with humans with a probability greater than 0.95.
R4	The rescue robot must avoid collisions with obstacles with a probability greater than 0.90.

the black-box predictive model. For all experiments, a neural network model for each requirement is embedded within the managing system to serve as a runtime predictor.

5.1 Evaluation subject

We consider the search and rescue robot illustrated in Section 2, and focus on a scenario where a rescue robot is navigating in a low-visibility environment to rescue injured persons. The camera mounted on the robot is trying to correctly classify the objects, such as obstacles and humans. Consider the following example. Due to the low-visibility environment, the robot's camera misclassified 20% of obstacles, which, in turn, increased the risk of collisions. Given these operating conditions, the predictive model anticipates the possible violation of R1. Thus, the analysis phase of the MAPE-K loop triggers the plan phase, which is responsible for the online opportunistic approach. The counterfactual is retrieved, suggesting a slight modification in the features, such as *cruise speed* from 4.5 m/s $\rightarrow$ 4.2 m/s and the *image resolution* from mid (720p) $\rightarrow$ high(1080p). The effect of this explainable adaptation is that the probability of classifying obstacles correctly increases from 80% $\rightarrow$ 87%, thus satisfying *R1*.

5.2 Testbed

All the experiments have been conducted on a commodity laptop running Windows 11 Home (Version 10.0, Build 22631). The hardware configuration includes an Intel Core i5-10300H processor (8 CPUs, 2.50 GHz) and 8 GB of RAM.

5.3 Results

Computational cost (RQ1) CSA-Φ and the selected baselines are executed with the initial configurations and the assignment of controllable features in the semantic space. Each of these initial configurations of features indicates the operating conditions for which adaptation is needed. In our evaluation, we have a total of 200 assignments of features. So, to answer RQ1, we calculate the execution time of CSA-Φ and the selected baselines. It is the time taken by the plan phase of the MAPE-K loop when the system is actually deployed in its environment. Once calculated, we compare the distribution of execution time taken by CSA-Φ and baselines.

Results: Figure 3 presents the execution time distribution of CSA-Φ and the two baseline methods using box plots (log scale). The results show that CSA-Φ exhibits

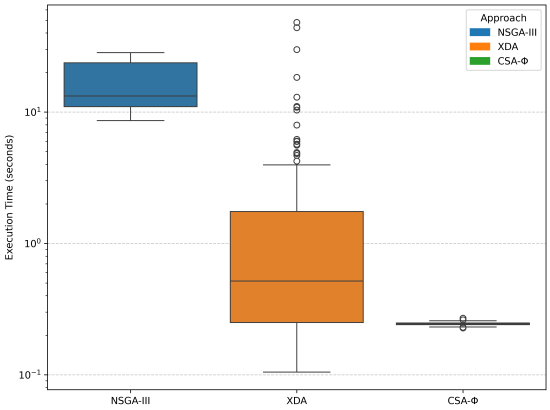

Fig. 3: Comparison of execution times in seconds (lower is better)

minimal variability, indicating consistent performance and computational efficiency. The median execution times for CSA-Φ, XDA, and NSGA-III are 0.24, 0.52, and 13.26 seconds, respectively. When compared with NSGA-III, XDA has a lower median execution time, but its performance shows high variability, as evidenced by multiple outliers. This suggests that XDA may incur significant computational overhead in certain cases, affecting its reliability in time-sensitive scenarios. To assess the statistical significance of CSA-Φ's execution time compared to the baselines, we conducted a Mann-Whitney U test [22], with the results summarized in Table 3. The extremely low p-values and large z-scores reported in the table indicate that the differences in execution time are statistically significant and unlikely to be due to random variation. White-box approaches like CSA-Φ and XDA are less computationally expensive than black-box methods such as NSGA-III, as shown in the evaluation results. Additionally, CSA-Φ exhibits a lower variance (i.e., a smaller interquartile range) compared to XDA, indicating more stable and consistent performance across different instances.

Table 3: Statistical significance for execution time

	p-value	**z-score**
XDA vs. CSA-Φ	2.45e-10	6.33
NSGA-III vs. CSA-Φ	2.56e-34	17.29

RQ1 summary: CSA-Φ demonstrates higher efficiency in execution time than baseline methods, consistently completing tasks faster with minimal variability. In contrast, NSGA-III incurs 87.25% higher computational cost on average.

Table 4: Statistical significance for satisfaction likelihood

	Requirements	p-value
XDA vs. CSA-Φ	R1	1.715e-1
	R2	6.190e-1
	R3	2.892e-2
	R4	2.883e-1
NSGA-III vs. CSA-Φ	R1	2.502e-1
	R2	3.289e-1
	R3	9.51e-3
	R4	4.469e-2

Effectiveness (RQ2) To address RQ2, we executed CSA-Φ using the same initial configurations and controllable feature assignments within the semantic space as used in RQ1. In total, 200 distinct feature assignments were evaluated to ensure statistical robustness and eliminate the possibility of results being influenced by chance. Additionally, we applied the Mann-Whitney U test to assess whether the observed differences in performance are statistically significant.

Two key metrics are used to evaluate the effectiveness of the approach: satisfaction likelihood, which measures the predicted probability that a requirement is satisfied under given operating conditions, and success rate, which quantifies how often the system successfully satisfies all requirements after adaptation.

Satisfaction likelihood is the probability, as given by the black box model, for the class label to be *true*. The mathematical formula for satisfaction likelihood is as follows:

$$SL(R_i) = P_{M_i}\Big(y = \text{true} \,|\, (\hat{CF}, \hat{OF})\Big)$$

where P_{M_i} is the predicted probability of the given black-box model M_i, for the class label y, which represents whether the requirement is satisfied; $\hat{CF}$ represents the adaptation decision applied to controllable features; and $\hat{OF}$ is the current state of the observable features.

Success rate is defined as the number of instances in X that satisfy the requirement R_i under the given operating conditions ($\hat{CF}$'$\hat{OF}$), such that the probability estimation of the class label to be *true* is greater than the threshold (which is set to 0.8). The mathematical formula for success rate is as follows:

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

where $n_{\text{success}}(R_i)$ indicates the number of successful cases in which the requirement R_i is satisfied. $|\cdot|$ represents cardinality (count), so in this context, it essentially means the number of elements in X.

Results: Figure 4 presents a box plot comparing the satisfaction likelihood (ranging from 0 to 1) across all system requirements. For requirement R2, CSA-Φ exhibits a wider distribution, indicating higher variance in satisfaction likelihood. In

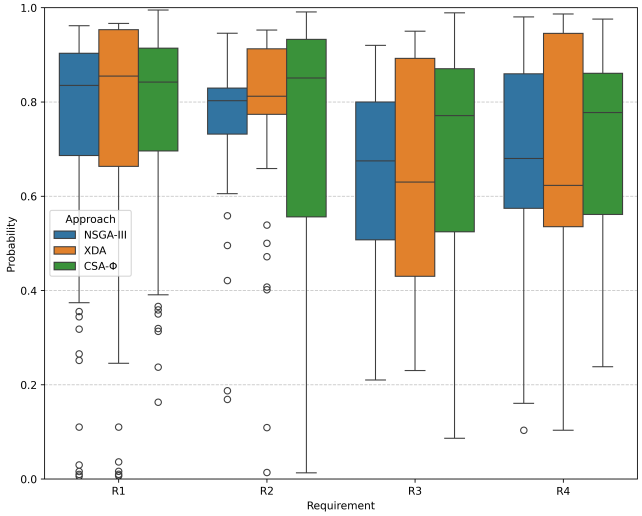

Fig. 4: Satisfaction likelihood (higher is better)

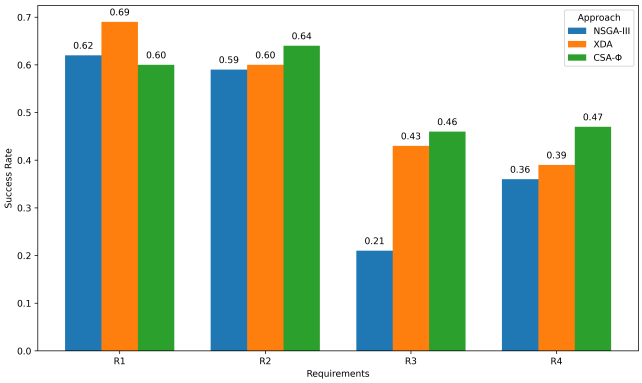

Fig. 5: Success rate (higher is better)

the cases of R1 and R2, all approaches display outliers below the lower whiskers, representing extreme data points that deviate from the general trend. Except for R1, the median satisfaction likelihood values achieved by CSA-Φ are slightly higher than those of XDA and NSGA-III. This suggests that CSA-Φ is at least as effective as traditional optimization methods in satisfying requirements, particularly when evaluated through the lens of satisfaction likelihood.

To assess the statistical significance of these results, we performed the Mann-Whitney U test. Table 4 summarizes the outcomes, comparing CSA-Φ against both XDA and NSGA-III. As shown in the table, statistical significance is not consistent across all requirements; however, it is evident in several cases, which are specifically highlighted. Figure 5 compares the success rate across all requirements for CSA-Φ and the baseline methods. As shown in the figure, CSA-Φ achieves a higher success rate in

most cases—specifically for requirements R2, R3, and R4—when compared to both NSGA-III and XDA. This demonstrates CSA-Φ 's consistency and effectiveness in fulfilling system requirements. The relative improvement over NSGA-III ranges from 0.032 to 0.543, highlighting a significant advantage in several scenarios. In comparison to XDA, the relative difference ranges from 0.063 to 0.170. These variations in relative success rates indicate that CSA-Φ provides substantial improvements over traditional black-box optimization techniques like NSGA-III and performs comparably (if not slightly better) than explanation-driven approaches such as XDA.

RQ2 summary:
CSA-Φ shows no statistically significant differences in satisfaction likelihood compared to baselines, though slightly higher medians suggest it is comparable to existing methods. It matches XDA in success rates while outperforming NSGA-III, demonstrating reliable adaptation compared to traditional black-box methods.

6 Threats to validity

One possible threat to internal validity arises from the counterfactual generation process, which may emphasize changing certain features while overlooking others. To address this, CSA-Φ selects features for modification based on confidence loss, rather than relying on random or arbitrary changes. This ensures that feature selection is guided by meaningful impact on the satisfaction of requirements.

To mitigate construct validity concerns, we compared CSA-Φ against two established baselines, ensuring that the evaluation is grounded in meaningful and relevant metrics.

Regarding external validity, our experiments were conducted in a simulated environment where full control over both controllable and observable features was assumed. This setup allows for consistent and repeatable evaluation. We also repeated each simulation 200 times to improve the robustness and generalizability of the results. Additionally, we used the Mann-Whitney U test to validate statistical significance, strengthening result reliability.

7 Related Work

Although the concept of explainability originated within the field of artificial intelligence, it has become an increasingly important non-functional quality when designing systems that rely on black-box models [1]. As AI components are increasingly integrated into software systems, explainability is now also gaining prominence in the domain of software engineering [26]. For instance, MAB-EX [4] enables systems to monitor their environment and internal behavior, identify situations requiring explanation, and construct human-understandable responses using internal models. This enhances the granularity of system explanations and the user collaboration through the integration of runtime models and reasoning capabilities.

Similarly, the Explainable Self-Adaptation (XSA) framework [7] aims to increase the transparency of self-adaptive systems by explicitly explaining adaptation decisions. Integrated into the feedback control loop, XSA improves both user trust and the engineer's ability to interpret and debug runtime decisions. Köhl et al. [19] further emphasized explainability as a non-functional requirement, highlighting the importance of systematically eliciting explainability requirements to guide the development of trustworthy systems.

The self-adaptive systems research community focuses on building systems capable of operating under uncertain conditions—or gracefully degrading when this is not possible. SafeX [20], for example, incorporates various explainable AI techniques to enhance safety and trust in autonomous systems. It supports real-time assessment and debugging by structuring explanations while preserving the reliability of AI components.

Weyns et al. [30] proposed a hybrid approach that integrates the MAPE loop, control theory, and machine learning to achieve robust adaptation. In this model, the MAPE loop governs high-level decision-making, control theory manages real-time system metrics, and machine learning contributes adaptive capabilities. Calinescu et al. [5] also underscored the importance of runtime learning, proactive adaptation, and formal guarantees to manage uncertainty in SAS effectively.

To support human-in-the-loop adaptation, stochastic models have been employed to explain system behavior under uncertainty [6]. In robotics, explanation-driven approaches are increasingly being adopted to help both engineers and end-users understand the rationale behind operational decisions, thus fostering trust [3].

8 Conclusion

In this paper, we introduced CSA-Φ, a novel approach that integrates model-agnostic counterfactual explanations into the feedback control loop of self-adaptive systems. By embedding explainability directly into the adaptation process, CSA-Φ provides meaningful insights that help steer adaptation decisions while reducing the computational cost typically associated with optimization in white-box models. We conducted an empirical evaluation to assess both the efficiency (measured by execution time) and effectiveness (measured by satisfaction likelihood and success rate) of CSA-Φ. The results demonstrate that CSA-Φ significantly outperforms the baselines in terms of execution time while achieving comparable performance in terms of satisfaction likelihood and success rate.

Acknowledgment

This work was supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF) and supported by the German Research Foundation (DFG) - SFB 1608 - 501798263 and KASTEL Security Research Labs, Karlsruhe.

References

1. Angelov, P.P., Soares, E.A., Jiang, R., Arnold, N.I., Atkinson, P.M.: Explainable artificial intelligence: an analytical review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **11**(5), e1424 (2021)
2. Bersani, M.M., Camilli, M., Lestingi, L., Mirandola, R., Rossi, M., Scandurra, P.: A conceptual framework for explainability requirements in software-intensive systems. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW). pp. 309–315 (2023)
3. Bersani, M.M., Camilli, M., Lestingi, L., Mirandola, R., Rossi, M., Scandurra, P.: Towards better trust in human-machine teaming through explainable dependability. In: 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C). pp. 86–90. IEEE (2023)
4. Blumreiter, M., Greenyer, J., et al.: Towards self-explainable cyber-physical systems. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 543–548. IEEE (2019)
5. Calinescu, R., Mirandola, R., Perez-Palacin, D., Weyns, D.: Understanding uncertainty in self-adaptive systems. In: IEEE international conference on autonomic computing and self-organizing systems (ACSOS). pp. 242–251 (2020)
6. Cámara, J., Silva, M., Garlan, D., Schmerl, B.: Explaining architectural design tradeoff spaces: A machine learning approach. In: European Conference on Software Architecture. pp. 49–65. Springer (2021)
7. Camilli, M., Mirandola, R., Scandurra, P.: Xsa: Explainable self-adaptation. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. pp. 1–5 (2022)
8. Camilli, M., Mirandola, R., Scandurra, P.: Enforcing resilience in cyber-physical systems via equilibrium verification at runtime. ACM Transactions on Autonomous and Adaptive Systems **18**(3), 1–32 (2023)
9. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE transactions on information theory **13**(1), 21–27 (1967)
10. De Lemos, R., et al.: Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany. pp. 1–32. Springer (2013)
11. Deb, K., Jain, H.: 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), 577–601 (2013)
12. Esfahani, N., Malek, S.: 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. pp. 214–238. Springer (2013)
13. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
14. Gheibi, O., Weyns, D.: Lifelong self-adaptation: Self-adaptation meets lifelong machine learning. In: Proceedings of the IEEE/ACM SEAMS. pp. 1–12 (2022)
15. Gheibi, O., Weyns, D., Quin, F.: Applying machine learning in self-adaptive systems: A systematic literature review. ACM TAAS **15**(3), 1–37 (2021)
16. Han, H., Wang, W.Y., Mao, B.H.: Borderline-smote: a new over-sampling method in imbalanced data sets learning. In: International conference on intelligent computing. pp. 878–887. Springer (2005)
17. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)

18. Khalid, N., Qureshi, N.A.: Towards self-explainable adaptive systems (seas): A requirements driven approach. In: REFSQ Workshops (2021)
19. Köhl, M.A., et al.: Explainability as a non-functional requirement. In: 2019 IEEE 27th international requirements engineering conference (RE). pp. 363–368. IEEE (2019)
20. Kuznietsov, A., Gyevnar, B., Wang, C., Peters, S., Albrecht, S.V.: Explainable ai for safe and trustworthy autonomous driving: a systematic review. IEEE Transactions on Intelligent Transportation Systems (2024)
21. Langford, M.A., Cheng, B.H.: "know what you know": Predicting behavior for learning-enabled systems when facing uncertainty. In: Proceedings of IEEE/ACM SEAMS. pp. 78–89 (2021)
22. Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. The annals of mathematical statistics pp. 50–60 (1947)
23. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artificial intelligence **267**, 1–38 (2019)
24. Molnar, C.: Interpretable machine learning. Lulu. com (2020)
25. Negri, F.R., Nicolosi, N., Camilli, M., Mirandola, R.: Explanation-driven self-adaptation using model-agnostic interpretable machine learning. In: Proceedings of IEEE/ACM SEAMS. pp. 189–199 (2024)
26. Tantithamthavorn, C.K., Jiarpakdee, J.: Explainable ai for software engineering. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 1–2. IEEE (2021)
27. Vapnik, V.N.: An overview of statistical learning theory. IEEE transactions on neural networks **10**(5), 988–999 (1999)
28. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: Automated decisions and the gdpr. Harv. JL & Tech. **31**, 841 (2017)
29. Weyns, D.: An introduction to self-adaptive systems: A contemporary software engineering perspective. John Wiley & Sons (2020)
30. Weyns, D., et al.: Towards better adaptive systems by combining mape, control theory, and machine learning. In: Proceedings of IEEE/ACM SEAMS. pp. 217–223 (2021)
31. Weyns, D., Gheibi, O., Quin, F., Van Der Donckt, J.: Deep learning for effective and efficient reduction of large adaptation spaces in self-adaptive systems. ACM TAAS **17**(1-2), 1–42 (2022)
32. Wohlrab, R., Cámara, J., Garlan, D., Schmerl, B.: Explaining quality attribute tradeoffs in automated planning for self-adaptive systems. Journal of Systems and Software **198**, 111538 (2023)