Self-adaptive Machine Learning Systems: Research Challenges and Opportunities

Chapter · August 2022

DOI: 10.1007/978-3-031-15116-3_7

6 authors, including:

Paolo Romano
University of Lisbon and INESC-ID, Lisbon, Portugal
183 PUBLICATIONS **1,944** CITATIONS
SEE PROFILE

David Garlan
Carnegie Mellon University
463 PUBLICATIONS **30,819** CITATIONS
SEE PROFILE

Gabriel A. Moreno
Carnegie Mellon University
51 PUBLICATIONS **1,407** CITATIONS
SEE PROFILE

Eunsuk Kang
Massachusetts Institute of Technology
78 PUBLICATIONS **1,242** CITATIONS
SEE PROFILE

Self-adaptative Machine Learning Systems: Research Challenges and Opportunities

Maria Casimiro[1,2], Paolo Romano[2], David Garlan[1], Gabriel A. Moreno[3], Eunsuk Kang[1], and Mark Klein[3]

[1] Institute for Software Research, Carnegie Mellon University, Pittsburgh, PA, USA
[2] INESC-ID, Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal
[3] Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract. Today's world is witnessing a shift from human-written software to machine-learned software, with the rise of systems that rely on machine learning. These systems typically operate in non-static environments, which are prone to unexpected changes, as is the case of self-driving cars and enterprise systems. In this context, machine-learned software can misbehave. Thus, it is paramount that these systems are capable of detecting problems with their machined-learned components and adapting themselves to maintain desired qualities. For instance, a fraud detection system that cannot adapt its machine-learned model to efficiently cope with emerging fraud patterns or changes in the volume of transactions is subject to losses of millions of dollars. In this paper, we take a first step towards the development of a framework for self-adaptation of systems that rely on machine-learned components. We describe: (i) a set of causes of machine-learned component misbehavior and a set of adaptation tactics inspired by the literature on machine learning, motivating them with the aid of two running examples from the enterprise systems and cyber-physical systems domains; (ii) the required changes to the MAPE-K loop, a popular control loop for self-adaptive systems; and (iii) the challenges associated with developing this framework. We conclude with a set of research questions to guide future work.

Keywords: Self-adaptive systems · Machine Learning · Model degradation · Learning-Enabled Systems · Learning-Enabled Components.

1 Introduction

In recent years, Learning-Enabled Systems (LES) have become ubiquitous in domains such as enterprise and cyber-physical systems. LES are composed of one or more machine-learning components, also known as Learning Enabled Components (LEC), whose behavior is derived from training data [40]. LECs are then embedded into a larger system containing traditional computational entities such as web services, databases, and operator interfaces. Examples include: fraud detection, which uses a classifier to detect fraudulent transactions [8]; medical diagnosis, which relies on Machine Learning (ML) for classifying types of diseases of patients [27]; self-driving cars, which use ML to determine whether

they should brake to avoid collision with moving objects (e.g., cars, pedestrians) [4, 44]; robots, which rely on ML models to predict the amount of remaining battery power [37]; targeted advertisement services, which rely on recommender systems to show users items that they may find interesting [20]; and smart homes/buildings, that rely on LECs for tasks such as face and voice recognition [63, 34] or occupancy prediction for proactive heating/cooling [28, 36].

Despite their widespread use, and similarly to non-machine learned components, LECs can fail to perform as expected [23, 40, 68], thereby reducing system utility. For instance, changes in a system's operating environment can introduce drifts in the input data of the ML components making them less accurate [57], or attacks may attempt to subvert the intended functionality of the system [33].

While the literature offers no immediate solution to guarantee the behavior of LES under changing environments [40], the large body of works on Self-Adaptive Systems (SAS) has already investigated a number of methods to tackle analogous problems for non-ML based systems. Specifically, SAS react to environment changes, faults and internal system issues to improve the system's behavior, utility and/or dependability [19]. These systems usually adopt an architecture based on the MAPE-K loop [38], which monitors the system, decides when adaptation is needed, selects the best course of action to improve the system, and executes it. In this field, the actions available for the system to execute are usually called *tactics*. This is an extensive and active research area that has made steady improvements for years and thus we propose the use of SAS to deal with environment changes and faults in the context of LESs and LECs.

In fact, there is a large number of emerging techniques that have been developed by the ML community for improving and correcting supervised ML models and that could in principle be used as adaptation tactics in a SAS. These range from off-line, full model retraining and replacement, at one extreme, to incremental approaches performed in-situ, at the other [58, 56, 35, 12, 46, 67].

Unfortunately, determining when and how to take advantage of such tactics to perform adaptation is far from being trivial. First, there is a large number of possible adaptation tactics that could potentially be applied to a ML component, but not all approaches work with all forms of ML models. For example, when a new family is moving into a new smart home, all the LECs should be fine-tuned to work as expected for that family [7]. Transfer learning [52] could be leveraged in this setting to speed up the process of fine-tuning these models. Yet, this tactic would yield the expected benefits only if the families are similar [49].

Second, the value of investing in improving the accuracy of a ML component is strongly context-dependent – often depending on both domain and timing considerations. For example, while a medical diagnosis system may support model retraining at run time, the latency of this tactic may make it infeasible for self-driving cars, which rely instead on swifter tactics (such as replacing the ML component entirely) to match real-time response requirements. In a different mode of operation, however, both types of tactics may be available, e.g., if the self-driving car is stopped (parked mode of operation), it may be feasible to retrain an under-performing model without compromising safety.

Third, calculating the costs and benefits of these tactics is difficult, particularly in a whole-system context, where improving a specific component's performance may or may not improve overall system utility. Costs include time, resources (processing, memory, power), and service disruption. Benefits derive for instance from increased accuracy or fairness of the ML component, which can in turn lead to better performing down-stream components and support overall business goals (e.g. by improving advertisement revenue). Yet, both costs [16, 45] and benefits [17] can be hard to quantify, thus making it challenging to reason about whether executing a ML adaptation tactic will improve system utility.

We argue that in order to harness the potential of the rich space of ML adaptation mechanisms, it is necessary to develop methods aimed to: i) identify which tactics, among the ones available to adapt an LES, are the most effective in a given context to maximize system utility, and ii) integrate them into modern adaptive systems architectures. Specifically, in this paper we attempt to bring some clarity to this emerging but critical aspect of SAS by outlining: **(i)** a set of causes of ML component performance degradation and a set of adaptation tactics derived from research on ML (Section 3), providing examples from the domains of enterprise systems (Section 4) and cyber-physical systems (Section 5); **(ii)** the architectural and algorithmic changes required to incorporate effective ML adaptation into the MAPE-K loop, a popular framework for monitoring and controlling self-adaptive systems (Section 6); and **(iii)** the modeling and engineering challenges associated with realizing the full potential for adaptation of LESs (Section 6). We conclude by introducing a set of open research questions and directions for future work. This work is an extended version of [14] which provides: **(i)** a more detailed and general discussion on the causes of ML component performance degradation; and **(ii)** a second use-case from the cyber-physical systems domain. This work further extends the original running example by mapping the detailed discussion on the causes of ML component performance degradation to examples in the enterprise systems domain.

2 Background & Related Work

Current literature on SAS focuses on managed systems that do not embed (nor rely upon) ML models [39]. That is, although the self-adaptation mechanism (i.e. managing system) may rely on ML to perform a given function (e.g., decide which tactic to execute), the actual system that is adapted (i.e. the managed system) does not rely on any ML component. These systems have at their disposal a set of tactics that, for instance, change a system's architecture (e.g., adding/removing servers) or the quality of the service they provide (e.g., increasing/decreasing the rendering quality of images) in response to environment changes [21]. Usually, tactic outcomes have some uncertainty that can be modeled via probabilistic methods given assumptions on the underlying hardware/software platforms and their characteristics. Further, one can measure the properties of such systems through the use of metrics such as latency, throughput and content quality.

Determining the costs and benefits of such adaptation tactics has been well researched and there are numerous techniques and algorithms to that end [25, 11, 30]. Yet, new challenges arise when considering managed systems that depend on LECs. Not only are we missing a well-understood and generally applicable set of tactics that SASs can use to adapt LESs, but also the properties of ML components, such as accuracy and fairness, may not change consistently with the tactic that is executed. For example, if we retrain a ML model, its accuracy is not always affected in the same way, but may depend on the samples available to retrain the model, on the duration of the retraining process, and on the model's hyper-parameters [16]. Similarly, model fairness may also be affected in different ways due to the training samples that are used during re-training [24].

To improve the self-adaptive capabilities of systems and their performance, recent research has proposed SASs that rely on ML techniques and models to adapt the system [32, 59]. Specifically, ML can be used in the adaptation manager to: update adaptation policies, predict resource usage, update run-time models, reduce adaptation spaces, predict anomalies, and collect knowledge. Further, learning can be leveraged to improve the Analysis and Plan components of the MAPE-K loop [32].

In this paper, we focus on the complementary problem of how to leverage self-adaptation to correct and adapt supervised ML components of a managed system. The goal is to increase overall utility of ML-based systems when their ML components are under-performing. This vision is aligned with the one presented by Bures [10] who claims that *"self-adaptation should stand in equal-to-equal relationship to AI. It should both benefit from AI and enable AI."* Extending this vision further, we argue that the techniques developed in this context could also be applied, in a recursive fashion, to self-adapt adaptation managers that rely on ML components to enhance their effectiveness and robustness. For instance, a planner that relies on ML to reduce the adaptation space could have its own self-adaptation manager to ensure that the ML component is working as expected.

Our vision ties in the field of self-adaptive systems with the field of life-long/continual learning [61, 41], which deals with open-world learning problems. In fact, dealing with open-world changes was identified [32] as an open problem in the SAS domain. Specifically, Lifelong Learning deals with the problem of leveraging past knowledge to learn a new task better and Continual Learning is focused on solving the problem of maintaining the accuracy of old tasks when learning new tasks [41]. The techniques developed in these fields can be leveraged by SASs to improve ML components when unexpected changes occur in the environment or when the performance of the ML component is degraded and affecting overall system utility. Our focus is on SASs and on how to integrate techniques from these research domains into a generic, yet rigorous/principled framework that can decide which ML component to adapt, how and when.

Finally, Gheibi et. al. [31] have recently proposed a framework that aims at enhancing the quality of ML-based adaptation managers in scenarios where the managed system is not ML-based. The proposed system reacts to concept drifts that lead the ML models to perform sub-optimally (hence hindering the

effectiveness of the adaptation manager) and automatically generates alternative ML-models (e.g., using different features) to replace the ones currently in use. Our framework targets a broader class of systems in which ML-components are part of the managed system and not only of the adaptation manager. Further, the conceptual framework presented in this work considers a broader set of adaptation tactics for ML-components, such as unlearning or having humans in the loop (see Section 3.2). The next section provides details on possible causes of ML component degradation and repair tactics inspired by this field of research.

3 Degradation and Repair of Learning-Enabled Components

Similarly to traditional components, LECs can fail, leading systems to undesirable states and to require repair of the under-performing components [40]. Thus, in this section we are interested in understanding what can lead an LEC to fail or produce erroneous outputs (Section 3.1), and what tactics are available to repair the components that are deteriorating system utility (Section 3.2). The different causes of degradation and the applicability of the tactics introduced in Section 3.2 will be exemplified in sections 4 and 5 using two use-cases from the fraud detection systems' and cyber-physical systems' domains, respectively.

3.1 Causes of Degradation of ML Components' Accuracy

ML approaches rely on a data-set composed of multi-dimensional input data and labels. Since this data-set is used for training the ML model, the environment from which these data-points are collected is usually known as the training environment. Then, once the ML model has been trained, it can be used by the system to make predictions in run-time. This is typically considered the testing environment as the model has never seen the current data-points. We will use the notions of training and testing environments throughout this section to introduce typical causes of degradation of ML components.

It is generally assumed that the distribution $p(y, \mathbf{x})$ of labels y and multi-dimensional input data $\mathbf{x}$ does not change between training and testing environments. However, when this assumption does not hold, and hence the prior distribution $P(\mathbf{x})$, or the posterior distribution $P(y)$, or any of the conditional distributions $P(y|\mathbf{x})$ or $P(\mathbf{x}|y)$ changes, one may be in the presence of a problem commonly known as data-set shift [56–58, 51]. This raises the question of whether the current model is still fit for the current environment. As recent work has shown, not all data-set shifts are malign [58]. Thus, an effective SAS should not only detect shifts, but also assess their actual impact on system utility.

The literature on ML has investigated several types of data-set shifts that have different characteristics. These different characteristics influence the impact that each type of shift has on a given system, and also how easy it is to deal with/detect the shift. Specifically, problems such as anomaly detection, novelty

detection, open set recognition, out of distribution detection, and outlier detection [70] are specific instances of the most common types of shift. We argue that the different types of shift are general enough to be representative of most of the issues addressed by the existing ML literature. The following paragraphs introduce these types of shift and give examples of typical sources of shift that can affect an LEC.

Covariate Shift. When the distribution of the inputs to a model changes, such that it becomes substantially different from the distribution on which the model was trained, we find ourselves in the presence of a problem commonly known as covariate shift. That is, the distribution $P(\mathbf{x})$ changes but the conditional distribution $P(y|\mathbf{x})$ remains the same. More formally, $P(\mathbf{x})_{train} \neq P(\mathbf{x})_{test}$ and $P(y|\mathbf{x})_{train} = P(y|\mathbf{x})_{test}$ [48]. This type of shift is usually analyzed to evaluate how a model generalizes and how robust it is when the feature space is altered at test time, i.e. while the system is executing.

Prior Probability Shift (label shift). Differently, when we are in the presence of prior probability shift, also known as label shift, this means that the distribution $P(y)$ of the labels/outputs has changed, i.e., the class proportions differ between training and test. More formally, $P(y)_{train} \neq P(y)_{test}$ and $P(\mathbf{x}|y)_{train} = P(\mathbf{x}|y)_{test}$ [48]. This can be seen as the inverse of covariate shift in the sense that now the distribution of features is the same between training and testing while the distribution of the labels changes. Dealing with this type of shift is particularly challenging when the new distribution $P(y)_{test}$ is unknown.

Concept Shift. Finally, concept shift corresponds to a change in the relationship between input and output distributions, although each remains the same. More formally, when this type of shift occurs, we can have $P(y|\mathbf{x})_{train} \neq P(y|\mathbf{x})_{test}$ and $P(\mathbf{x})_{train} = P(\mathbf{x})_{test}$ or $P(\mathbf{x}|y)_{train} \neq P(\mathbf{x}|y)_{test}$ and $P(y)_{train} = P(y)_{test}$ [48].

Although these are the most common types of shift, it is also possible that other types of shift occur. We list them for completeness but give examples only of the most common ones in the remainder of the paper. Other types of shift that can happen are for instance when both the conditional distribution and the features/labels distribution changes. Formally, this would correspond to $P(y|\mathbf{x})_{train} \neq P(y|\mathbf{x})_{test}$ and $P(\mathbf{x})_{train} \neq P(\mathbf{x})_{test}$ or $P(\mathbf{x}|y)_{train} \neq P(\mathbf{x}|y)_{test}$ and $P(y)_{train} \neq P(y)_{test}$ [48]. These types of shift are typically less investigated in the literature since they are not so common in real world applications and also because they are extremely difficult to detect and deal with.

Sources of Data Shift. During the normal operation of a system, shift in the data can occur due to several reasons: to the passing of time, incorrect data or sample selection bias. Next, we provide details on each of these sources of shift.

Natural Drift due to Time. An effect of the natural passing of time is that people's tastes and behavior patterns change [7, 43]. For example, due to the passing of time and due to inflation, the value of money decreases. A static LEC, that is never adapted and does not account for these natural changes will gradually start producing worse predictions.

Incorrect Data. This problem arises when there are samples in the model's training set that are incorrectly labeled [66] or when test data is tampered with, thus leading the model to mispredict for inputs with specific characteristics. The former can happen for instance when unsupervised techniques are used to label examples in order to bootstrap the training set of a second supervised model [66]. Incorrect data can also make their way into a model's training set due to attackers that intentionally pollute it (e.g., by maliciously altering some of the input features) so as to cause the ML component to incorrectly predict outputs for certain inputs [35, 33]. Finally, noise and uncertainty, due to sensor errors or due to errors from upstream components, may also change the input data to an LEC, possibly causing drift and mispredictions.

Sample Selection Bias. This occurs when selecting data-points for a training set or when performing data cleaning [4]. While selecting data-points, there may be environmental factors that cause some inputs or labels to be sampled more often. For example, when selecting participants for a survey, steps must be taken to ensure that the population of interest is accurately represented. Similarly, when performing data cleaning, for example for a digit recognition task, less clear digits may be thrown away. However, this may prevent the model from learning that some digits are intrinsically harder to write than others [57]. During these, arguably critical, phases of model construction, sample selection bias will cause the training distribution to follow a different distribution than the test distribution, hence causing data shift and potentially a drop in system utility.

3.2 Repair Tactics

Table 1 illustrates a collection of tactics that can be used to deal with issues introduced by ML-based components caused by the different types of shift previously introduced. These tactics were inspired by research on ML [61, 46, 52, 12, 67]. Next, we describe the tactics presented in the table, discussing their costs and benefits, and motivating them in the following sections with scenarios from enterprise systems (Section 4) and cyber-physical systems domains (Section 5).

Component replacement. This tactic assumes the existence of a repository of components and respective meta-data that can be analyzed to determine if there exists a component that is better suited for the current system state, i.e., that is expected to lead to a higher system utility. If such a component exists, then this

[4] In ML, data cleaning corresponds to the process of identifying and correcting errors in a dataset that may negatively impact a predictive model.

Table 1. Examples of general adaptation tactics for ML-based systems with their strengths ('+') and weaknesses ('−').

Tactic	Description	Properties
Component Replacement	Replace an under-performing component by one that better matches the current environment	+ Fast and inexpensive, when possible − Alternative components may not be available in all scenarios − Alternative estimators, when available, may be more robust but less precise
Human-based Labeling [46]	Rely on a human to classify incoming samples or to correct the labeling of samples in the training set	+ Accuracy of human-based labels expected to be high − Expert knowledge may be expensive to obtain and/or introduce unacceptable latency
Transfer Learning [52]	Reuse knowledge gathered previously on different tasks/problems to accelerate the learning of new tasks	+ Less data-hungry than plain retrain − Effectiveness dependent on the similarities between old and new tasks/data − Computationally intensive process
Unlearning [12]	Remove samples that are no longer representative from the training set and from the model	+ Fast when ratio between data to forget and data-set size is small − Cost/latency for identifying examples to unlearn can be large and context-dependent
Retrain [67]	Retrain with new data and maybe choose new values for the ML model's hyper-parameters	+ Generic and robust method − Computationally intensive process − Accuracy and latency of the retrain process may vary significantly − Effective only once a relatively large number of instances of the new data are available

tactic will replace the under-performing component by the one that is expected to be better. A benefit of this tactic, whenever it is available, is to enable a swift reaction to data-set shifts. Its main cost depends on the latency and resources used for the analysis of the candidate components available in the repository. Additionally, alternative components may not always be available and, when they do exist, they may be less precise albeit more robust.

Human-based labeling. Humans are often able to recognize patterns, problems, and objects more accurately than ML components [46]. Thus, depending on the domain, humans may play a role in correcting these components or giving them correct samples [46, 65]. For example, when an LEC is highly uncertain about a specific input, it may rely on a human to provide a label. Similarly, if incorrect data is found on a model's data-set, a human may be asked to correct those samples. While this tactic may provide high benefit if the human is an expert, it also has a high cost, since humans are expensive. Also, if there is a significant amount of samples to label, the latency of the process may be unacceptable.

Transfer learning. Transfer learning (TL) techniques leverage knowledge obtained when performing previous tasks that are similar to the current one so that learning the current task becomes easier [52, 42]. For this tactic to be applicable, it is necessary to evaluate the similarity between the source and target

tasks/domains. Transferring knowledge between dissimilar tasks will not provide benefits. In order to compute this similarity, metrics such as LEEP [49] can be used. The advantages of this tactic are that it requires less data than is needed to retrain a model from scratch, thus allowing for a quicker model initialization phase. However, the process of TL, similarly to a model retrain, is also computationally intensive.

Unlearning. This tactic corresponds to unlearning data that no longer reflects the current environment/state of the system and its lineage, thus eliminating the effect of that data on current predictions [12], while avoiding a full model retrain. A key problem that stands in the way of the execution of this tactic is the identification of incorrect labels. In scenarios in which the identification of incorrect samples is not readily available, one may leverage automatic techniques, such as the one described in [13], which are faster but typically less accurate than relying on humans. As such, the cost and complexity of this adaptation tactic vary depending on the context. Then, after identifying the incorrect samples, the model must be updated to accurately reflect the correct data. The advantage of unlearning techniques with respect to a typical full model retrain is the time savings (up to several orders of magnitude [12]) that can be achieved.

Retrain and/or hyper-parameter optimization. This is a general tactic that involves retraining the model with new data that reflects recent relevant data-set shifts. There are many types of retraining, ranging from a simple model refresh (incorporate new data using old hyper-parameters), to a full retrain (including hyper-parameter optimization, possibly encompassing the search for different model types/architectures [26]). These imply different computational costs and lead to different benefits in terms of model accuracy improvements. In the presence of data-set shifts, when there is new data that already incorporates the new input distribution, this tactic often represents a simple, yet possibly expensive, approach to deal with this problem. However, this tactic usually requires a substantial amount of data to yield highly accurate models and is computationally intensive. The benefits of this tactic are dependent on the type of retrain process and on the quality of the new data. As for its cost, if retraining is performed on the cloud, it can be directly converted to the economic cost of the virtual machines. Several techniques exist to predict such costs [2, 69, 16, 45].

4 Adaptation of ML-based Enterprise Systems

We now motivate the need for self-adaptive Learning-Enabled Systems through an example from the enterprise systems domain. We provide examples of situations in which each type of shift can occur as well as of scenarios in which each repair tactic can be applied.

4.1 Running Example – Fraud Detection System

Consider a fraud detection system that relies on ML models for determining whether credit/debit card transactions are legitimate or fraudulent. These ML

models typically attribute a score to each transaction, which corresponds to the likelihood of the transaction being fraudulent [56]. The score attributed by the ML model is then used by a rule-based model to decide whether transactions are legitimate or fraudulent. Typical clients of companies that provide fraud detection services are banks and merchants. In this setting, system utility is typically defined based on attributes such as the cost of losing clients due to incorrectly declined transactions, fairness (no user sees its transactions declined more often than others) [24] and the overall cost of service level agreement (SLA) violations (these systems have strict SLAs to process transactions in real time, e.g. at most 200ms on the 99.999th percentile of the latencies' distribution [8]).

While cost and revenue are directly affected by the ML model's mispredictions, response time is affected by model complexity, i.e., more complex models may introduce higher latencies that compromise SLAs. Thus, when adapting an LEC in this domain, it is necessary to account for the impact of increased complexity on the fulfillment of the SLAs. Further, the impact of LEC mispredictions varies not only from client to client, with whom different SLAs may have been agreed upon, but also in time, since during specific periods, e.g., Black Friday, the volume of transactions is substantially increased. During busy days such as these, since there is an increase in the number of legitimate transactions and the spending patterns are altered, it is crucial that ML models are less strict and reduce false alarms. At the same time, having less strict models may lead to more fraudulent transactions being accepted. Hence, mispredictions come at huge penalties and a delicate tradeoff is required to ensure an acceptable system utility. Finally, these systems are subject to constantly evolving fraud patterns, to which the ML components must adapt [3].

4.2 Causes of Degradation of ML Components' Accuracy

This section illustrates each type of data-shift with examples from the fraud detection systems domain.

Covariate Shift. In a fraud detection system, covariate shift occurs when patterns of legitimate transactions change, for instance due to busy shopping days like Black Friday and Christmas [3]. Although the actual features used for classification may not change, their distribution does. For example, suppose a user usually purchases items online from shop **A**. The distribution of fraud given the feature *online shop* is 10% for shop **A**. The user then discovers that shop **B** actually sells the same items but at a cheaper price, so they start purchasing from shop **B**. In such a setting, the distribution of the feature *online shop*, given by $P(\mathbf{x})$ is altered, but the distribution of fraud given the feature, $P(y|\mathbf{x})$, remains the same, i.e., there is the same amount of fraud in shops **A** and **B**, regardless of where the user buys. This scenario will possibly lead the fraud detection system to suspect that the change in the user's behavior is actually fraud because it learned that the user typically buys from shop **A**.

As an example of incorrect data leading to shifts, security breaches could lead attackers to poison the data used for training ML models, hence causing them to make incorrect predictions.

Prior Probability Shift (label shift). In the context of fraud detection systems, this type of shift occurs for example when the proportion of fraudulent transactions in the training set is different than for the test set [43], i.e., $P(y)_{train} \neq P(y)_{test}$. This type of shift requires assuming that the distribution of input data given fraud, $P(\mathbf{x}|y)$, does not change between training and testing environments. This corresponds to a mathematical abstraction over the power of an adversary that is capable of generating fraudulent transactions that follow the same pattern as legitimate transactions. Since the model has learned the typical distribution of fraud, it's predictions will follow that distribution, which is no longer representative of the system's environment.

Concept Shift. This is the most common type of data-set shift in the fraud detection domain and occurs when fraudsters adapt their strategies and new fraud patterns emerge, such that the ML model is no longer able to effectively distinguish fraudulent from legitimate transactions [43]. In this case, it is the distribution $P(y|\mathbf{x})$ that changes while $P(\mathbf{x})$ remains the same.

4.3 Repair Tactics

This section motivates the applicability of each repair tactic by providing examples of their usage when different causes of degradation have affected the system's LECs.

Component replacement. When the volume of transactions changes, for instance during special days such as Black Friday, ML models may consider the increased frequency of transactions as an indicator of fraud and erroneously flag legitimate transactions as fraudulent. Such mispredictions can lead to significant financial losses [8], thus requiring timely fixes that render the use of high latency tactics infeasible (note that in this context transactions need to be accepted/rejected within a few hundreds milliseconds [8]). As such, only low latency tactics can be applied. An example is to replace the under-performing models with rule-based models, e.g., developed by experts for specific situations, and/or to switch to previously trained models that are known to perform well in similar conditions.

Human-based labeling. Whenever the ML component suspects a transaction of being fraudulent it can automatically block that transaction. Then, the user can be informed of the decision and asked whether the transaction should be authorized or declined in the future. Another possibility is to add humans to the loop when adding samples to the ML component's training set. In this scenario, an expert can be asked to review the most uncertain classifications so as to improve the quality of the training samples. In the former scenario, the benefits are easily quantifiable, since the risk of accepting a possibly fraudulent transaction can

be measured via its economic value. However, users may get annoyed if their transactions are canceled too often, to the extent that they may stop purchasing using that credit card provider. As for relying on experts to review uncertain classifications, having an on-demand expert performing this task is expensive and the latency of the manual labeling process may be unacceptable [5].

Transfer learning. Suppose that: (i) a fraud detection company has a set of clients (such as banks), (ii) the company has a unique ML model for each client, so that it complies with data privacy regulations, and (iii) one of its clients is affected by a new attack pattern, which is eventually learned by that client's model. In this scenario, TL techniques can be used to improve other clients' models so that they can react to the same attack pattern. In fact, since privacy is important in this domain, there are techniques that can be used to deal with the problem of ensuring data confidentiality and anonymity in information transfer between clients [42, 29] instead of typical TL techniques that do not provide this assurance [52]. Estimating the benefits of executing this tactic for a given client boils down to estimating the likelihood that this client may be targeted by the same attack, which comes at an added cost and time. Yet, the execution of this tactic typically implies high computational costs (e.g., if cloud resources are used) and non-negligible latency, which may render this tactic economically unfavorable, or even inadequate, e.g., if the attack on a different client is imminent and the TL process is slow.

Unlearning. In the domain of fraud detection, if after a specific amount of time (e.g. 1 month) the fraud detection system does not receive complaints about a set of transactions, these will be labeled as legitimate. However, since users typically take a long time reviewing their statements and complaining when they do not recognize some transactions, it is possible that there are incorrectly labeled transactions in the data-set. In this scenario, and if a model has been trained with the incorrect samples, it is possible to leverage this tactic to remove the incorrect samples from the model without requiring a full model retrain.

Retrain and/or hyper-parameter optimization. Full model retraining can be leveraged for example when there is a new fraud pattern for which there is already enough data. By retraining the model with this data, the model is likely to increase the amount of fraudulent transactions detected, thus also increasing system utility. However, as this is a slow tactic, while it executes system utility is likely to either drop or remain as unsatisfactory as it was prior to the execution of the tactic. To prevent such situations, hybrid planning approaches can be leveraged [53] to execute a swift tactic that slightly improves system utility while the slow tactic is executing.

[5] Fraud detection systems normally rely on a fixed set of humans at any given time. This determines a maximum load of transactions that can be processed with a human in the loop.

5 Adaptation of ML-based Cyber-Physical Systems

Turning now to a different domain, in the context of learning-enabled cyber-physical systems (CPS) [55, 6, 18, 4] self-adaptation of its constituent LECs can also be seen as a mechanism through which system utility can be maintained. Thus, in this section we: present a motivating example from this domain; exemplify how the causes of degradation of ML component's accuracy presented in the previous section could occur in this context; and give examples of settings in which each adaptation tactic could be applied to improve overall system utility.

5.1 Running Example – Smart Homes

As a second example, consider a smart home that relies on ML to perform tasks such as face recognition for home security [63]; voice recognition for home entertainment [34]; occupancy prediction for proactive heating, cooling, lighting [36, 60, 28]. For each of these components, different system utility definitions are possible. For example in the case of an occupancy prediction LES, system utility could be defined in terms of user thermal comfort: if the temperature in the house is set according to the user's preferences, system utility will be high. In this scenario, and in situations in which the number of inhabitants changes drastically, for example due to Covid-19 that forced a significant number of people to stay home, the occupancy predictor can make incorrect predictions that will cause discomfort to users. In such a setting, adapting the occupancy predictor could improve user comfort and thus maintain system utility at a desired level.

5.2 Causes of Degradation of ML Components' Accuracy

Similarly to the enterprise systems domain, in the CPS domain one must also analyze the possible causes of ML mispredictions. As such, we now provide examples of each type of ML misprediction for the smart home example.

Covariate Shift. An example of covariate shift in the context of smart homes consists of noise/uncertainty in sensors that measure air quality. This noise changes the distribution of features that go into the predictive model possibly leading a ML classifier to mispredict air quality and thus mispredict the need to ventilate a room. Noise and uncertainty could also affect smart meters used to reduce energy consumption. Finally, different types of light (night versus day, dusk versus dawn) illuminating faces may lead a face recognition system to mispredict whether someone is an intruder. This example is an instance of covariate shift since the distribution of the features is altered due to the different types of light but the distribution of labels is the same and the conditional distribution of labels given the features also remains the same. It is also possible, in both domains (CPS and enterprise systems), that faults in the input data fed by some system component (LEC or non-LEC) to an LEC lead to mispredictions. This highlights the need for a system-wide perspective that considers both LEC and non-LEC aspects of the system.

As an example of LEC mispredictions in the CPS domain due to incorrect data, the literature on adversarial ML has shown how if a data point has been adversarially manipulated, a classification model using smart-meter data as input may not correctly identify which appliances are functioning in a smart home, thus not being able to properly reduce energy consumption [62].

Prior Probability Shift (label shift). Voice controllers in smart homes serve the purpose of executing commands issued by users. In such a setting, a voice controller in a smart home is subject to label shift when a command which was very rarely used is now used very often. Since the voice controller does not account for a change in its frequency of use, it will often mispredict the required action to execute when the command is issued. In this situation, the features and the relationship between features and labels is the same, but the actual correct class that should be derived from those features has changed, thus leading to an error.

Concept Shift. Smart homes rely on models that predict inhabitants' activity patterns. When these patterns are altered [7] this corresponds to concept shift. This can happen for instance due to big life events such as the birth of a child, adoption of a new pet, or having visitors stay over for a few days. When the patterns change, the features that are fed to the model do not change, neither do the labels. The only change is the relation between the two distributions which can cause the LEC to incorrectly predict the need for different tasks/settings.

5.3 Repair Tactics

This section illustrates in which scenarios each of the previously introduced adaptation tactics could be applied in the context of smart homes.

Component replacement. This tactic could be applied to enhance system utility in cases when the face recognition system behaves poorly for example due to low lighting at night, or due to sun rays illuminating faces at different angles at dusk or dawn. In such situations, the face recognition LEC could be replaced by a different component (LEC or non-LEC) that is known to perform better under the current environmental conditions. A benefit of this tactic, for example when compared with a retrain tactic that could potentially be applied in the same situation, is a quick improvement of system utility due to a speedy replacement of the under-performing component by one that is guaranteed to achieve a minimum desired system utility.

Human-based labeling. When adversaries have manipulated smart meters, such that the appliances currently working in a home cannot be properly detected, a human can be queried to clarify which appliances are working, thus improving the accuracy of the LEC and enabling an increase of system utility by contributing to reducing energy consumption.

Transfer learning. This tactic can be applied to bootstrap LECs of houses/rooms with new inhabitants [7]. Specifically, since different homes can have different configurations of sensors, rooms, and occupancy, the LECs will require fine-tuning not only to each user but also to each house. In this setting, TL techniques (e.g., based on multi-task Bayesian Optimization [64]), can exploit knowledge gathered from homes with similar configurations and whose LECs have already been optimized.

Unlearning. This tactic could be applied for example when adversaries change smart meter data. In this context, this tactic could be applied to forget these incorrect data points so as not to pollute the LEC's training set. Similarly, this tactic could also be applied when the behaviors of the inhabitants have changed. In such a situation, unlearning old behaviors may be a suitable tactic to prevent mispredictions. However, the benefits of applying this tactic are dependent on the amount of data that needs to be forgotten: in case there are plenty of examples to forget, retraining may actually be faster at achieving the same results [12].

Retrain and/or hyper-parameter optimization. In a smart home, the tactic of retraining an LEC could be available in a self-adaptive LES repertoire to deal with scenarios such as when inhabitants have new routines, which may cause occupancy prediction LECs to misbehave. In this situation, the LEC can be retrained on new examples that represent these new routines, thus improving the quality of its predictions and, ultimately, user satisfaction and thermal comfort. Similarly, whenever a voice recognition system fails to recognize a voice or a control, it is possible to retrain the model with examples of the voice/control such that it learns to predict them. This tactic can also be applied in settings in which modifying the model structure also yields better predictions. For example, for face recognition systems used for security purposes, which have more stringent deadlines and require higher accuracies, a retrain tactic could train a new type of model to replace the old one, ensuring that the new model is faster (e.g., replacing neural networks by decision trees) or train a set of models to increase confidence in the predictions [9].

Table 2 summarizes the examples of situations provided in the previous sections that can occur in each domain (enterprise systems and cyber-physical systems). Each situation is an instance of each type of shift, and each tactic exemplifies how to deal with the shift in the different situations.

6 MAPE-K Loop for Learning-Enabled Systems

In SAS, the MAPE-K loop typically actuates over a system composed of traditional components, i.e., non-LEC components. However, as illustrated in Figure 1, LESs generally encompass both non-LEC and LEC components. We argue that the MAPE-K loop should be revised in order to be able to cope with the

Table 2. Examples of causes of ML misbehavior within each domain — enterprise systems (ES) and cyber-physical systems (CPS) — and tactics available for adaptation in each scenario.

Problem	Domain	Example Situation	Applicable Tactics
Covariate shift	ES	Transaction patterns change Adversaries poison data	• Component replacement • Unlearning
	CPS	Noise/uncertainty in sensors	• Transfer learning
		Different lighting conditions for face recognition LEC	• Component replacement
		Adversaries manipulate smart meter data	• Human-based labeling • Unlearning
Label shift	ES	Variable fraud rate	• Human-based labeling
	CPS	Unknown command for voice controller	• Human-based labeling
Concept shift	ES	New fraud strategies	• Transfer learning
	CPS	Inhabitant's living patterns	• Retrain • Unlearning

unique issues described in Section 3.1 that affect LECs by effectively leveraging the adaptation tactics presented in Section 3.2.

In the following, we discuss the new research challenges and opportunities that arise in each of the MAPE-K loop stages due to the LEC specific adaptation tactics.

6.1 Monitor

The *Monitor* stage has to keep track of the inputs received by the ML components because shifts of the input distributions may affect the predictions. For instance, the detection of out-of-distribution inputs may mean that there has been a change in the environment and thus the model used by some ML component may no longer be representative of the current environment. The challenge here is not only detecting the occurrence of shifts in a timely and reliable fashion, but also how to effectively characterize them — since different types of shifts require different reaction methods.

As in other SAS, typical attributes that contribute to the system's utility (e.g., latency, throughput) or the satisfaction of required system properties must be monitored. In addition to these, the *Monitor* stage must also gather the outputs of the ML component to account for situations in which changes in the inputs go by unnoticed, perhaps because they are too slow, but that manifest themselves faster in the outputs [72]. Examples of outputs to monitor are, for instance, shifts in the output distribution, model's accuracy and error — obtained by comparing predictions with real outcomes. A relevant challenge here is that often real outcomes are only known after a long time, if ever. For instance, in fraud detection, false negatives (i.e., undetected real fraud) are known only when users file a complaint. Approaches such as those proposed in [72, 56, 71] provide a good starting point for the implementation of a *Monitor* for self-adaptive learning-enabled systems.

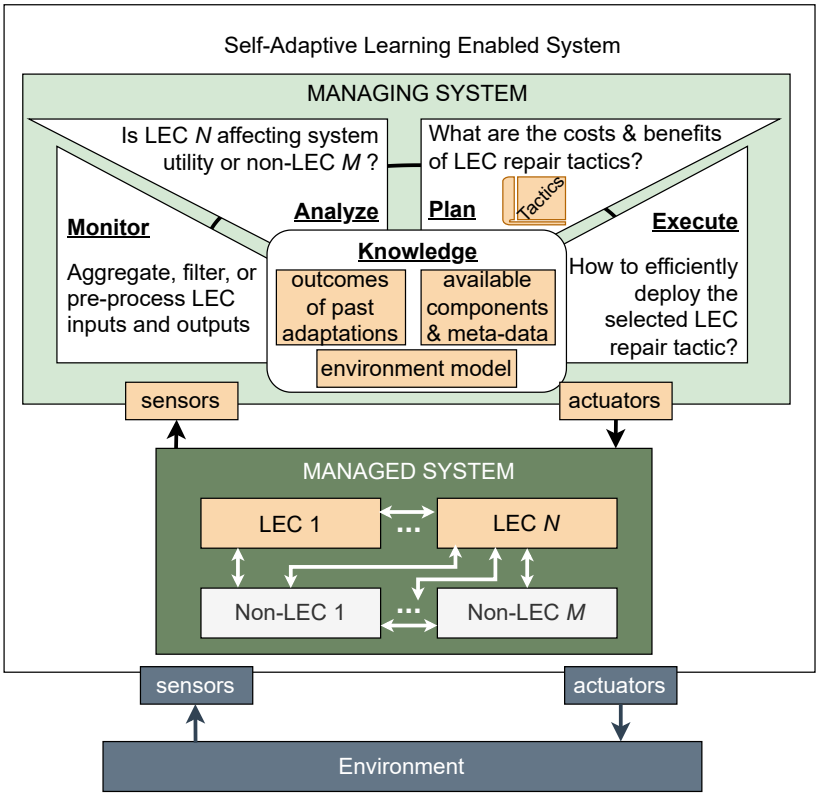

Fig. 1. MAPE-K loop over an LES with a mix of LECs and components that are not learning-enabled, with specific challenges for each MAPE-K stage. White arrows represent dependencies between components.

Challenges. Monitoring input and output distributions requires keeping track of a multitude of features and parameters, which would otherwise be disregarded. This is already challenging due to the amount of data that needs to be stored, maintained, and analyzed. Finding suitable frequencies to gather these data and adapting them in the face of evolving time constraints is an even bigger challenge in time-critical domains [5, 56].

6.2 Analyze

The *Analyze* stage is responsible for determining whether degradations of the prediction quality of LECs are affecting (or are predicted to affect) other system components and system utility to such an extent that adaptation may be required. To accomplish this, one can leverage techniques developed by the ML community to detect possible issues in the inputs and outputs of the LEC [56–58, 72], errors in its training set [1] and the appearance of new features relevant for

prediction [54]. These techniques must then be adjusted for each system, which includes adapting them to different ML models and tasks.

Challenges. Estimating the impact of an LEC on other system components and on system utility can be challenging because often (mis)predictions affect the system's utility/dependability in ways that are not only application- but also context-dependent. For instance, during periods with higher transaction volumes, such as on Black Friday, mispredictions have higher impact on system utility, since during these periods it is more critical to accurately detect fraud, while maximizing accepted transactions. Architectural models can capture the information flows among components, but the challenge is to estimate how the uncertainty in the output of the LECs propagates throughout the system.

6.3 Plan

The *Plan* stage is responsible for identifying which adaptation tactics (if any) to employ to address issues with LECs affecting the system. As with other self-adaptation approaches, this reasoning should consider the costs and benefits of each viable tactic. Further, most of the proposed tactics have a non-negligible latency, which needs to be accounted for as in latency-aware approaches [47]. An additional concern is that some of these tactics may require a considerable use of resources to execute, either in the system itself or offloaded. This requires *Plan* to account for this impact or cost. For LESs that rely on multiple LECs, whenever a system property is (expected to be) violated or when system utility decreases, fault localization may be required to understand which component is under-performing and should be repaired/replaced [22].

Challenges. Although there are several approaches [2, 16, 69] that attempt to predict the time/cost of training ML models, this is a complex problem that is strongly influenced by the type of ML models considered, their hyper-parameters and the underlying (cloud) infrastructure used for training. These techniques represent a natural starting point to estimate the costs and benefits of adaptation tactics such as the ones presented. Yet, developing techniques for predicting the costs/benefits of complex tactics, e.g., unlearning, remains an open challenge.

One interesting direction is to exploit techniques for estimating the uncertainty [51] of ML models to quantify both the likelihood of models' mispredictions as well as the potential benefits deriving from employing corrective adaptation tactics. While some ML models can directly estimate their own uncertainty [50], others require additional techniques (e.g. ensembles [9]) to obtain uncertainty estimations. Still, existing techniques can suffer from significant shortcomings in practical settings [51].

Finally, tactics that modify LECs are typically computationally expensive (e.g., non-negligible latency). Thus, *Plan* must have mechanisms to verify that the system can execute the tactic without compromising other components/properties, or even the entire system.

6.4 Execute

To execute a given adaptation tactic, the *Execute* stage must have access to mechanisms to improve or replace the LEC and/or its training set. As in the conventional MAPE-K loop, we require implementations of adaptation tactics that are not only efficient to execute, but also have predictable costs/benefits and are resilient to run-time exceptions.

Challenges. A key challenge is how to enhance the predictability of the execution of the ML adaptation tactics, which often require the processing of large volumes of data (e.g., to re-train a large scale model) possibly under stringent timing constraints. We argue that the community of SAS would benefit from the availability of open-source software frameworks that implement a range of generic adaptation tactics for LECs. These frameworks would allow to mask complexity, promote interoperability and comparability of SAS. Further, it would also provide an opportunity to assemble, in a common framework, techniques that have been proposed over many years in different areas of the AI/ML literature.

6.5 Knowledge

Finally, the *Knowledge* module is responsible for maintaining information that reflects what is known about the environment and the system. As in traditional systems, in the case of self-adaptive LESs *Knowledge* also needs to maintain information about the environment so that trends can be observed. These trends can be crucial to detect the shifts that may lead to mispredictions. Additionally, for LESs, the *Knowledge* component should evolve in order to keep track of the costs/benefits of each tactic on the affected LECs and system's utility. This corresponds to gathering: knowledge on how each tactic altered an LEC and on the context in which the tactic was executed; and meta information on training sets, for instance characterizing the most important features for predicting the costs and benefits of the different tactics. This added knowledge should be leveraged to improve the decision making process and thus improve adaptation. By gathering knowledge on how each tactic altered an LEC and on the context in which the tactic was executed, the *Analyze* and *Plan* stages can take more effective decisions on when to adapt and which tactic to execute, respectively. Finally, for a tactic that replaces under-performing LECs, *Knowledge* must contain a repository of the available components and their meta-data. This meta-data, we argue, should provide information to enable reasoning on whether the necessary preconditions to enable a safe and timely reconfiguration hold.

7 Conclusions and Future Work

This work introduced a vision for a new breed of self-adaptive frameworks that brings together techniques developed by the ML literature (used here as adaptation tactics), and reasons about the cost/benefit tradeoffs of each, with the end goal of adapting LECs of learning-enabled systems to maintain or improve

system utility. With the aid of two running examples we showed how different adaptation tactics can be applied to repair LECs when different real-life situations hinder system utility.

Further, we identified a set of key requirements that should be supported by the various elements of the classic MAPE-K control loop and a set of challenging research problems, such as: **(i)** How to estimate the costs and benefits of each tactic? **(ii)** How to reason about the impact of LEC mispredictions on system utility? **(iii)** How to determine whether changes to one LEC impact other components in the system? **(iv)** How to reason about the long-term impacts of adaptation tactics on system utility?

We have started to address these research questions by developing a framework to reason about the costs and benefits of executing a repair tactic [15]. This framework relies on probabilistic model checking to determine whether repairing an LEC at a specific point in time will improve overall system utility in the future. As next steps, we plan to study the benefits of executing retrain tactics in the context of a fraud detection system. We aim to develop simple models that are able to estimate the benefits of retrain within acceptable bounds. These models can then be integrated into the model checking framework to determine when to adapt the fraud detection system.

Acknowledgements

Support for this research was provided by Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) through the Carnegie Mellon Portugal Program under Grant SFRH/BD/150643/2020 and via projects with references POCI-01-0247-FEDER-045915, POCI-01-0247-FEDER-045907, and UIDB/50021/2020. The contributions of Gabriel Moreno and Mark Klein are based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. Such contributions are used with permission, but ownership of the underlying intellectual property embodied within such contributions is retained by Carnegie Mellon University. DM22-0149

References

1. Abedjan, Z., et al.: Detecting data errors: Where are we and what needs to be done? Procs. of VLDB **9**(12) (2016)
2. Alipourfard, O., et al.: Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: Procs. of NSDI (2017)
3. Aparício, D., et al.: Arms: Automated rules management system for fraud detection. arXiv preprint arXiv:2002.06075 (2020)
4. Badue, C., Guidolini, R., et al.: Self-driving cars: A survey. Expert Systems with Applications **165**, 113816 (2021)

5. Bartocci, E., et al.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In: Lectures on Runtime Verification. Springer (2018)
6. Bojarski, M., et al.: End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316 (2016)
7. Bouchabou, D., Nguyen, S.M., Lohr, C., LeDuc, B., Kanellos, I.: A survey of human activity recognition in smart homes based on iot sensors algorithms: Taxonomies, challenges, and opportunities with deep learning. Sensors **21**(18) (2021)
8. Branco, B., et al.: Interleaved sequence rnns for fraud detection. In: Procs. of KDD (2020)
9. Breiman, L.: Bagging predictors. In: Machine Learning. vol. 24. Springer (1996)
10. Bureš, T.: Self-adaptation 2.0. In: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) (2021)
11. Cámara, J., Lopes, A., Garlan, D., Schmerl, B.: Adaptation impact and environment models for architecture-based self-adaptive systems. Science of Computer Programming **127**, 50–75 (2016)
12. Cao, Y., Yang, J.: Towards making systems forget with machine unlearning. In: Procs. of S&P. IEEE (2015)
13. Cao, Y., et al.: Efficient repair of polluted machine learning systems via causal unlearning. In: Procs. of Asia CCS (2018)
14. Casimiro, M., Romano, P., Garlan, D., Moreno, G., Kang, E., Klein, M.: Self-adaptation for machine learning based systems. In: Procs. of SAML. LNCS, Springer (2021)
15. Casimiro, M., Garlan, D., Cámara, J., Rodrigues, L., Romano, P.: A probabilistic model checking approach to self-adapting machine learning systems. In: Procs. of ASYDE, Co-located with SEFM 2021 (2021)
16. Casimiro, M., et al.: Lynceus: Cost-efficient tuning and provisioning of data analytic jobs. In: Procs. of ICDCS (2020)
17. Chen, T.: All versus one: An empirical comparison on retrained and incremental machine learning for modeling performance of adaptable software. In: Procs. of SEAMS. IEEE (2019)
18. Chen, Z., Huang, X.: End-to-end learning for lane keeping of self-driving cars. In: Procs. of IV (2017)
19. Cheng, B.H.C., et al.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. Springer (2009)
20. Cheng, H.T., et al.: Wide & deep learning for recommender systems. In: Procs. of DLRS (2016)
21. Cheng, S.W., et al.: Evaluating the effectiveness of the rainbow self-adaptive system. In: Procs. of SEAMS. IEEE (2009)
22. Christi, A., et al.: Evaluating fault localization for resource adaptation via test-based software modification. In: Procs. of QRS (2019)
23. Cito, J., Dillig, I., Kim, S., Murali, V., Chandra, S.: Explaining mispredictions of machine learning models using rule induction. In: Procs. of ESEC/FSE (2021)
24. Cruz, A.F., et al.: A bandit-based algorithm for fairness-aware hyperparameter optimization. CoRR **abs/2010.03665** (2020)
25. deGrandis, P., Valetto, G.: Elicitation and utilization of application-level utility functions. In: Procs. of ICAC (2009)
26. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. The Journal of Machine Learning Research **20**(1), 1997–2017 (2019)
27. Erickson, B.J., et al.: Machine learning for medical imaging. Radiographics **37**(2) (2017)

28. Esrafilian-Najafabadi, M., Haghighat, F.: Occupancy-based hvac control systems in buildings: A state-of-the-art review. Building and Environment **197**, 107810 (2021)
29. Gao, D., Liu, Y., Huang, A., Ju, C., Yu, H., Yang, Q.: Privacy-preserving heterogeneous federated transfer learning. In: 2019 IEEE International Conference on Big Data (Big Data). pp. 2552–2559. IEEE (2019)
30. Ghahremani, S., Giese, H., Vogel, T.: Improving scalability and reward of utility-driven self-healing for large dynamic architectures. ACM Trans. Auton. Adapt. Syst. **14**(3) (feb 2020)
31. Gheibi, O., Weyns, D.: Lifelong self-adaptation: Self-adaptation meets lifelong machine learning. In: Procs. of SEAMS (2022)
32. Gheibi, O., et al.: Applying machine learning in self-adaptive systems: A systematic literature review. arXiv preprint arXiv:2103.04112 (2021)
33. Gu, T., et al.: Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access **7** (2019)
34. Guo, X., Shen, Z., Zhang, Y., Wu, T.: Review on the application of artificial intelligence in smart homes. Smart Cities **2**(3), 402–420 (2019)
35. Huang, L., et al.: Adversarial machine learning. In: Procs. of AISec (2011)
36. Huchuk, B., Sanner, S., O'Brien, W.: Comparison of machine learning models for occupancy prediction in residential buildings using connected thermostat data. Building and Environment **160**, 106177 (2019)
37. Jamshidi, P., et al.: Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots. In: Procs. of SEAMS (2019)
38. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003)
39. Krupitzer, C., et al.: A survey on engineering approaches for self-adaptive systems (2018)
40. Langford, M.A., Chan, K.H., Fleck, J.E., McKinley, P.K., Cheng, B.H.: Modalas: Model-driven assurance for learning-enabled autonomous systems. In: Procs. of MODELS (2021)
41. Liu, B.: Learning on the job: Online lifelong and continual learning. In: Procs. of the AAAI Conference on Artificial Intelligence. vol. 34 (2020)
42. Liu, Y., et al.: A secure federated transfer learning framework. Procs. of IS **35**(4) (2020)
43. Lucas, Y., Jurgovsky, J.: Credit card fraud detection using machine learning: A survey. CoRR **abs/2010.06479** (2020)
44. Mallozzi, P., Pelliccione, P., Knauss, A., Berger, C., Mohammadiha, N.: Autonomous vehicles: state of the art, future trends, and challenges. Automotive Systems and Software Engineering pp. 347–367 (2019)
45. Mendes, P., et al.: TrimTuner: Efficient optimization of machine learning jobs in the cloud via sub-sampling. In: MASCOTS (2020)
46. Miller, B., et al.: Reviewer integration and performance measurement for malware detection. In: Procs. of DIMVA (2016)
47. Moreno, G.A., et al.: Flexible and efficient decision-making for proactive latency-aware self-adaptation. ACM Trans. Auton. Adapt. Syst. **13**(1) (2018)
48. Moreno-Torres, J.G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N.V., Herrera, F.: A unifying view on dataset shift in classification. Pattern recognition **45**(1), 521–530 (2012)
49. Nguyen, C., Hassner, T., Seeger, M., Archambeau, C.: Leep: A new measure to evaluate transferability of learned representations. In: Procs. of ICML. PMLR (2020)

50. Osborne, M.A., et al.: Gaussian processes for global optimization. In: LION (2009)
51. Ovadia, Y., et al.: Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In: Procs. of NIPS (2019)
52. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE TKDE **22**(10) (2009)
53. Pandey, A., Moreno, G.A., Cámara, J., Garlan, D.: Hybrid planning for decision making in self-adaptive systems. In: Procs. of SASO (2016)
54. Papamartzivanos, D., et al.: Introducing deep learning self-adaptive misuse network intrusion detection systems. IEEE Access **7** (2019)
55. Peng, Z., Yang, J., Chen, T.H., Ma, L.: A first look at the integration of machine learning models in complex autonomous driving systems: a case study on apollo. In: Procs. of ESEC/FSE (2020)
56. Pinto, F., et al.: Automatic model monitoring for data streams. arXiv preprint arXiv:1908.04240 (2019)
57. Quionero-Candela, J., et al.: Dataset shift in machine learning. The MIT Press (2009)
58. Rabanser, S., et al.: Failing loudly: An empirical study of methods for detecting dataset shift. In: Procs. of NIPS (2019)
59. Saputri, T.R.D., Lee, S.W.: The application of machine learning in self-adaptive systems: A systematic literature review. IEEE Access **8** (2020)
60. Shi, J., Yu, N., Yao, W.: Energy efficient building hvac control algorithm with real-time occupancy prediction. Energy Procedia **111**, 267–276 (2017)
61. Silver, D.L., Yang, Q., Li, L.: Lifelong machine learning systems: Beyond learning algorithms. In: 2013 AAAI spring symposium series (2013)
62. Singh, A., Sikdar, B.: Adversarial attack for deep learning based iot appliance classification techniques. In: 2021 IEEE 7th World Forum on Internet of Things (WF-IoT). IEEE (2021)
63. Surantha, N., Wicaksono, W.R.: Design of smart home security system using object recognition and pir sensor. Procedia computer science **135**, 465–472 (2018)
64. Swersky, K., et al.: Multi-task bayesian optimization. Procs. of NIPS **26** (2013)
65. Wang, Z.J., Choi, D., Xu, S., Yang, D.: Putting humans in the natural language processing loop: A survey. arXiv preprint arXiv:2103.04044 (2021)
66. Wu, D., et al.: A highly accurate framework for self-labeled semisupervised classification in industrial applications. IEEE TII **14**(3) (2018)
67. Wu, Y., et al.: DeltaGrad: Rapid retraining of machine learning models. In: Procs. of ICML (2020)
68. Xiao, Y., Beschastnikh, I., Rosenblum, D.S., Sun, C., Elbaum, S., Lin, Y., Dong, J.S.: Self-checking deep neural networks in deployment. In: Procs. of ICSE (2021)
69. Yadwadkar, N.J., Hariharan, B., Gonzalez, J.E., Smith, B., Katz, R.H.: Selecting the ¡i¿best¡/i¿ vm across multiple public clouds: A data-driven performance modeling approach. In: Procs. of SoCC. p. 452–465 (2017)
70. Yang, J., Zhou, K., Li, Y., Liu, Z.: Generalized out-of-distribution detection: A survey. arXiv preprint arXiv:2110.11334 (2021)
71. Yang, Z., Asyrofi, M.H., Lo, D.: BiasRV: Uncovering biased sentiment predictions at runtime. CoRR **abs/2105.14874** (2021)
72. Zhou, X., et al.: A Framework to Monitor Machine Learning Systems Using Concept Drift Detection. Springer (2019)