Self-Adaptation for Machine Learning Based Systems

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 adapt 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 aimed to self-adapt 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 a running example; (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 the paper with a set of research questions to guide future work.

Keywords

Self-adaptive systems, Machine Learning, Model degradation

1. Introduction

The field of self-adaptive systems (SAS) is an extensive and active research area that has made steady improvements for years. SAS react to environment changes, faults and internal system issues to improve the system's behavior, utility and/or dependability [1]. These systems usually adopt an architecture, known as the MAPE-K loop, which monitors the system, decides when it needs adaptation, selects the best course of action to improve the system, and executes it [2]. The actions available for the system to execute are usually called *tactics*. The literature on SAS spans a broad range of systems such as enterprise systems, and cyber-physical systems (CPS).

In parallel with the maturing of SAS research, a new class of systems has emerged: supervised and semi-supervised machine learning (ML) based systems are now becoming ubiquitous. Such systems embed one or more components, whose behavior is derived from training data, into a larger system containing traditional computational entities (web services, databases, operator interfaces). Examples include: fraud detection, which uses a classifier to detect fraudulent transactions [3]; medical diagnosis, which relies on ML for classifying types of diseases of sick patients [4]; self-driving cars, which use ML to determine whether they should stop based on how distant they are from the car in front [5]; robots, which rely on ML models to predict the amount of remaining battery power [6]; and targeted advertisement services, which rely on recommender systems to show users items that they may find interesting [7].

For such systems, adaptation poses a key concern. In addition to the reasons that traditional systems must adapt (faults, changing requirements, unexpected loads, etc.), ML-based components may fail to perform as expected, thereby reducing system utility. For instance, changes in a system's operating environment can introduce drifts in the input data of the ML models making them less accurate [8], or attacks may attempt to subvert the intended functionality of the system [9].

Thankfully, there is a large number of emerging techniques that have been developed by the ML community for adapting supervised ML models and that could in principle be used as adaptation tactics in a self-adaptive system. These range from off-line, from-scratch model retraining and replacement, at one extreme, to incremental approaches performed in-situ, at the other [10, 11, 12, 13, 14, 15]. And more techniques are being developed constantly.

Unfortunately, determining when and how to take advantage of such tactics to perform adaptation is highly non-trivial. First, there is a large number of possible adaptation tactics that could potentially be applied to an ML component, but not all approaches work with

SAML'21: International Workshop on Software Architecture and Machine Learning, September 13–17, 2021, Växjö, Sweden
maria.casimiro@tecnico.ulisboa.pt (M. Casimiro); romano@inesc-id.pt (P. Romano); garlan@cs.cmu.edu (D. Garlan); gmoreno@sei.cmu.edu (G. A. Moreno); eunsukk@andrew.cmu.edu (E. Kang); mk@sei.cmu.edu (M. Klein)
© 2021 Copyright for this paper by Carnegie Mellon University and the authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

all forms of supervised ML models. For example, some training models may allow a system to selectively "forget" certain inputs, while others do not. Similarly, some ML models support transfer learning to incrementally update a learnt model, but not all do.

Second, the value of investing in improving the accuracy of an ML component is strongly context-dependent – often depending on both the 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) that can address real-time system 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 underperforming model without compromising safety.

Third, calculating the costs and benefits of these tactics is difficult, particularly in a whole-system context, where improving a particular 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). Both costs and benefits can be hard to quantify, however, and hence to reason about when determining whether an ML adaptation tactic makes sense.

We argue, therefore, that in order to harness the potential of the rich space of ML adaptation mechanisms, it is necessary to develop methods that can reason about which tactics are available to adapt the ML component, which are the most effective to employ in a given context so that system utility is maximized, and how to 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 (§ 3); **(ii)** 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 (§ 4); and **(iii)** the modeling and engineering challenges associated with realizing the full potential for adaptation of ML-based systems (§ 4). We conclude with a set of open research questions.

2. Background & Related Work

Current literature on SAS focuses on managed systems that do not embed (nor rely upon) ML models [16]. That is, although the self-adaptation mechanism (i.e. managing system) may rely on ML to perform a given function (e.g., decide the 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. 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 for that end [17]. However, new challenges arise when considering managed systems that depend on ML models. Not only are we missing a well-understood and generally applicable set of tactics that SAS can use to adapt ML-based systems, 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 an 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. Similarly, model fairness may also be affected in different ways due to the training samples that are fed during re-training [18].

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 [19, 20]. Specifically, ML is used in the adaptation manager to: update adaptation policies, predict resource usage, update run-time models, reduce adaptation spaces, predict anomalies, and collect knowledge. Additionally, learning is typically leveraged to improve the Analysis and Plan components of the MAPE-K loop [19].

In this paper, we focus on the problem of how to leverage self-adaptation to correct and adapt supervised ML components of a managed system, while increasing overall utility of ML-based systems when their ML components are underperforming. This vision is aligned with the one presented by Bures [21] in which the author 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.

The vision presented in this paper differs from work on collective SAS since we are targeting systems with only one agent and with a centralized learning process, whereas this line of research focuses on systems with multiple agents that can share knowledge with each other.

Differently, our vision ties in the field of lifelong/continual learning [22, 23], which deals with open-world problems, with the field of self-adaptive systems. In fact, dealing with open-world changes was identified by Gheibi et. al. [19] 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 [23]. The techniques developed in this domain 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 affects overall system utility. Overall, 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. The next section provides details on possible causes of ML component degradation and repair tactics inspired by this field of research.

3. Adaptation of ML-based Systems

We now motivate the need for self-adaptive ML-based systems through an example from the enterprise systems domain. Then, we present a set of possible causes for ML component performance degradation and a set of adaptation tactics.

3.1. Running Example – Fraud Detection System.

Consider a fraud detection system that relies on ML models for scoring credit/debit card transactions. 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 client is declined more often) [18] 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 [3]). While cost and revenue are directly affected by ML model's mispredictions, response time is affected by model complexity, i.e., more complex models may introduce higher latencies that compromise SLAs. However, the impact of these 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 altered. During busy days such as these, adapting the ML models responsible for fraud detection so that they are less strict and reduce false alarms is crucial in order to preserve system utility. However, this adaptation entails a delicate trade-off, since less strict models can allow fraudulent transactions to be accepted. Further, these systems are subject to constantly evolving fraud patterns, to which the ML models must adapt [24].

3.2. Causes of Degradation of ML Components' Accuracy

We now focus on problems that deteriorate the performance of ML components such that they are no longer able to maintain system utility at a desired level. In particular, we present two classes of problems, which, we argue, are general enough to be representative of most of the issues addressed by the existing ML literature.

Data-set 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 data-set shift [8, 11, 10, 25]. As recent work has shown, not all data-set shifts are malign [10]. As such, an effective SAS should not only detect shifts, but also be able to assess their actual impact on system utility. In a fraud detection system, data-set shift occurs when new fraud patterns emerge (e.g., charges at a particular merchant), or when patterns of legitimate transactions change, for instance due to busy shopping days like Black Friday and Christmas [24]. Although the actual features used for classification may not change, their distribution does. This means that different values of the features now characterize legitimate and fraudulent transactions.

Incorrect Data. This problem arises when there are samples in the model's training set that are incorrectly labeled [26] or when test data is tampered with, thus leading the model to mispredict when certain inputs arrive. 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 [26]. Incorrect data can also make their way into a model's training set due to attackers that intentionally pollute it so as to cause the ML component to incorrectly predict outputs for certain inputs [12, 9]. For instance, in the fraud detection case, security breaches could lead to

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 – Non ML-based estimators may not be available in all scenarios – Alternative estimators, when available, may be more robust but less precise
Human-based Labeling [14]	Rely on a human to classify some 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 [27]	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 [13]	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 [15]	Retrain with new data and maybe choose new values for the ML model's hyper-parameters	+ Generic and robust method – Effective only once a relatively large number of instances of the new data are available – Computationally intensive process – Accuracy and latency of the retrain process may vary significantly

poisoning the data used for training ML models, hence causing them to make incorrect predictions.

3.3. Repair Tactics

Table 1 illustrates a collection of tactics that can be used to deal with issues introduced by ML-based components. These tactics were inspired by research on ML [22, 14, 27, 13, 15]. Next, we describe the tactics presented in the table, motivating them with scenarios in which they can be applied and discussing their costs and benefits.

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. For example, when the volume of transactions changes, for instance in 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 [3], thus requiring timely fixes and rendering the use of high latency tactics infeasible (note that in this context, transactions need to be accepted/rejected within milliseconds [3]). As such, only low latency tactics can be applied. An example is to replace the underperforming 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. A benefit of this tactic, whenever it is available, is too enable a swift reaction to data set shifts. Its main cost depends on the latency and resources used for the analysis of the candidate replacing components available in the repository.

Human-based labeling. Humans are often able to recognize patterns, problems, and objects more accurately than ML components [14]. Thus, depending on the domain, humans may play a role in correcting these components or giving them correct samples [14]. For instance, whenever the ML component suspects a transaction of being fraudulent, it can be automatically canceled. 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.

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 [27]. 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[1] , 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 [29, 27] can be used to improve the other clients' models so that they can react to the same attack. Estimating the benefits of executing this tactic for a given client boils down to estimating the likelihood that this client may suffer the same attack. Yet, the execution of this tactic typically implies high computational costs (e.g., if cloud resources are used)

[1]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 [28].

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. 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 [13], 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. For instance, in a fraud detection system, incorrectly classified transactions may all be eventually identified for "free", although with large latencies, when users review their credit card statements. Conversely, in scenarios in which the identification of incorrect samples is not readily available, one may leverage automatic techniques, such as the one described in [30], which are faster but typically less accurate. As such, the cost and complexity of this task vary depending on the context. Then, after identifying the incorrect samples, the model must be updated to accurately reflect the correct data. At this point, the advantage of unlearning techniques with respect to a typical full model retrain is the time savings (up to 9.5×10^4) that can be achieved [13].

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 drifts, e.g., a new kind of attack in a fraud detection system. 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 different model types/architectures), which imply different computational costs and can benefit model's accuracy at different extents. In the presence of data-set shift, 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. 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 renting the virtual machines and several techniques exist to predict such costs [31, 32].

4. MAPE-K Loop for ML-Based Systems

In SAS, the MAPE-K loop typically actuates over a system composed of non-ML components. To enable the development of self-adaptive ML-based systems, in which the MAPE-K loop actuates over a system composed of non-ML and ML components (Figure 1) we argue that each stage of the MAPE-K loop should be revised to effectively leverage tactics such as the ones mentioned.

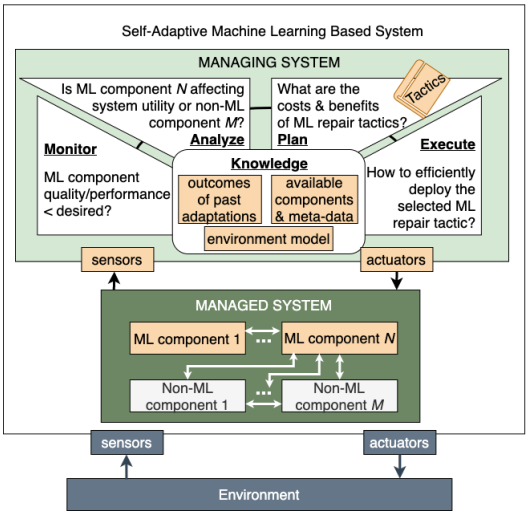


Figure 1: MAPE-K loop over an ML-based system with a mix of ML and non-ML components, with specific challenges for each MAPE-K stage. White arrows represent dependencies between components.

4.1. Monitor

The *Monitor* stage has to keep track of the inputs used when querying 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 [33]. 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 and false positives are normally undetectable (since no feedback is obtained for transactions that are legitimate but rejected by the system). Approaches such as those proposed in [33, 11, 34] provide a good starting point for the implementation of a *Monitor* for self-adaptive ML-based systems.

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 [35, 11].

4.2. Analyze

The *Analyze* stage is responsible for determining whether degradations of the prediction quality of ML components are affecting (or 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 model [8, 11, 10, 33], errors in its training set [36] and the appearance of new features relevant for prediction [37]. These techniques must then be adjusted for the particular case of each system, which includes adapting them to different ML models and tasks.

Challenges. Estimating the impact of an ML component 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 ML components propagates throughout the system.

4.3. Plan

The *Plan* stage is responsible for identifying which adaptation tactics (if any) to employ to address issues with ML components 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[38]. 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 ML-based systems that rely on multiple ML components, whenever a system property is (expected to be) violated or when system utility decreases, fault localization may be required to understand which component is underperforming and should be repaired/replaced [39].

Challenges. Although there are several approaches [31, 40] 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. 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 [25] of ML models to quantify both the likelihood of models' mispredictions as well as the potential benefits deriving from employing corrective adaptation tactics. Certain ML models can directly estimate their own uncertainty [41], or additional techniques (e.g. ensembles [42]) can be used to obtain uncertainty estimations. Still, existing techniques can suffer from significant shortcomings in practical settings [25].

Finally, tactics that modify ML components are 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.

4.4. Execute

To execute a given adaptation tactic, the *Execute* stage must have access to mechanisms to improve or replace the ML component 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 ML components.

This would allow one 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.

4.5. Knowledge

Finally, the *Knowledge* module is responsible for maintaining information that reflects what is known about the environment and the system. For ML-based systems, the *Knowledge* component should evolve in order to keep track of the costs/benefits of each tactic on the affected ML components and system's utility. This corresponds to: gathering knowledge on how each tactic altered an ML component 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 ML component 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 underperforming ML components with non ML-based ones, *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.

5. 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/benefits trade offs of each, with the end goal of adapting degraded ML components of ML-based systems to maintain system utility. With the aid of a running example we showed how different adaptation tactics can be applied to repair ML models 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. Finally, we highlight the following research questions as directions for future work: **(i)** How to estimate the costs and benefits of each tactic? **(ii)** How to reason about the impact of ML mispredictions on system utility? **(iii)** How do changes to one ML component impact the other components in the system? **(iv)** How to reason about the long-term impacts of adaptation tactics on system utility?

Acknowledgments

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. This material is 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. DM21-0052

References

[1] B. H. C. Cheng, et al., Software Engineering for Self-Adaptive Systems: A Research Roadmap, Springer, 2009.

[2] J. O. Kephart, D. M. Chess, The vision of autonomic computing, Computer 36 (2003).

[3] B. Branco, et al., Interleaved sequence rnns for fraud detection, in: Procs. of KDD, 2020.

[4] B. J. Erickson, et al., Machine learning for medical imaging, Radiographics 37 (2017).

[5] Z. Chen, X. Huang, End-to-end learning for lane keeping of self-driving cars, in: Procs. of IV, 2017.

[6] P. Jamshidi, et al., Machine learning meets quantitative planning: Enabling self-adaptation in autonomous robots, in: Procs. of SEAMS, 2019.

[7] H.-T. Cheng, et al., Wide & deep learning for recommender systems, in: Procs. of DLRS, 2016.

[8] J. Quionero-Candela, et al., Dataset shift in machine learning, The MIT Press, 2009.

[9] T. Gu, et al., Badnets: Evaluating backdooring attacks on deep neural networks, IEEE Access 7 (2019).

[10] S. Rabanser, et al., Failing loudly: An empirical study of methods for detecting dataset shift, in: Procs. of NIPS, 2019.

[11] F. Pinto, et al., Automatic model monitoring for data streams, arXiv preprint arXiv:1908.04240 (2019).

[12] L. Huang, et al., Adversarial machine learning, in: Procs. of AISec, 2011.

[13] Y. Cao, J. Yang, Towards making systems forget with machine unlearning, in: Procs. of S&P, IEEE, 2015.

[14] B. Miller, et al., Reviewer integration and performance measurement for malware detection, in: Procs. of DIMVA, 2016.

[15] Y. Wu, et al., DeltaGrad: Rapid retraining of machine learning models, in: Procs. of ICML, 2020.

[16] C. Krupitzer, et al., A survey on engineering approaches for self-adaptive systems (2018).
[17] K. Ervasti, A survey on network measurement: Concepts, techniques, and tools (2016).
[18] A. F. Cruz, et al., A bandit-based algorithm for fairness-aware hyperparameter optimization, CoRR abs/2010.03665 (2020).
[19] O. Gheibi, et al., Applying machine learning in self-adaptive systems: A systematic literature review, arXiv preprint arXiv:2103.04112 (2021).
[20] T. R. D. Saputri, S.-W. Lee, The application of machine learning in self-adaptive systems: A systematic literature review, IEEE Access 8 (2020).
[21] T. Bureš, Self-adaptation 2.0, in: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2021.
[22] D. L. Silver, Q. Yang, L. Li, Lifelong machine learning systems: Beyond learning algorithms, in: 2013 AAAI spring symposium series, 2013.
[23] B. Liu, Learning on the job: Online lifelong and continual learning, in: Procs. of the AAAI Conference on Artificial Intelligence, volume 34, 2020.
[24] D. Aparício, et al., Arms: Automated rules management system for fraud detection, arXiv preprint arXiv:2002.06075 (2020).
[25] Y. Ovadia, et al., Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift, in: Procs. of NIPS, 2019.
[26] D. Wu, et al., A highly accurate framework for self-labeled semisupervised classification in industrial applications, IEEE TII 14 (2018).
[27] S. J. Pan, Q. Yang, A survey on transfer learning, IEEE TKDE 22 (2009).
[28] Y. Liu, et al., A secure federated transfer learning framework, Procs. of IS 35 (2020).
[29] K. Swersky, et al., Multi-task bayesian optimization, Procs. of NIPS 26 (2013).
[30] Y. Cao, et al., Efficient repair of polluted machine learning systems via causal unlearning, in: Procs. of Asia CCS, 2018.
[31] M. Casimiro, et al., Lynceus: Cost-efficient tuning and provisioning of data analytic jobs, in: Procs. of ICDCS, 2020.
[32] P. Mendes, et al., TrimTuner: Efficient optimization of machine learning jobs in the cloud via subsampling, in: MASCOTS, 2020.
[33] X. Zhou, et al., A Framework to Monitor Machine Learning Systems Using Concept Drift Detection, Springer, 2019.
[34] Z. Yang, M. H. Asyrofi, D. Lo, BiasRV: Uncovering biased sentiment predictions at runtime, CoRR abs/2105.14874 (2021). arXiv:2105.14874.
[35] E. Bartocci, et al., Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications, in: Lectures on Runtime Verification, Springer, 2018.
[36] Z. Abedjan, et al., Detecting data errors: Where are we and what needs to be done?, Procs. of VLDB 9 (2016).
[37] D. Papamartzivanos, et al., Introducing deep learning self-adaptive misuse network intrusion detection systems, IEEE Access 7 (2019).
[38] G. A. Moreno, et al., Flexible and efficient decision-making for proactive latency-aware self-adaptation, ACM Trans. Auton. Adapt. Syst. 13 (2018).
[39] A. Christi, et al., Evaluating fault localization for resource adaptation via test-based software modification, in: Procs. of QRS, 2019.
[40] O. Alipourfard, et al., Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics, in: Procs. of NSDI, 2017.
[41] M. A. Osborne, et al., Gaussian processes for global optimization, in: LION, 2009.
[42] L. Breiman, Bagging predictors, in: Machine Learning, volume 24, Springer, 1996.