Evolvability of Machine Learning-based Systems: An Architectural Design Decision Framework

Joran Leest*, Ilias Gerostathopoulos*, Claudia Raibulet*†
*Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
Email: {j.g.leest, i.g.gerostathopoulos, c.raibulet}@vu.nl
†Universita' degli Studi di Milano-Bicocca, Milan, Italy
Email: claudia.raibulet@unimib.it

***Abstract*—The increasing integration of machine learning (ML) in modern software systems has lead to new challenges as a result of the shift from human-determined behavior to data-determined behavior. One of the key relevant challenges concerns *concept drift* (CD), i.e., the potential performance degradation due to changes in the data distribution. CD may severely affect the quality of the provided services, being also difficult to predict and detect, as well as costly to address. In this context, we focus on the *evolvability* of ML-based systems and the architectural considerations in addressing this concern. In this paper, we propose a novel scenario-based framework to support, justify and underpin architectural design decisions that address evolvability concerns in ML-based systems. The applicability and relevance of our framework is outlined through an illustrative example. We envision our framework to be extended to address other quality attributes important to ML-based systems and, overall, provide architectural support for ML operations (MLOps). Finally, we outline our plan to apply it in a number of industrial case studies, evaluate it with practitioners, and iteratively refine it.**

***Index Terms*—software architecture, machine learning, design decisions, quality attributes, evolvability, concept drift**

I. Introduction

A major challenge of software engineering nowadays is related to serving, monitoring, and evolving machine learning (ML) models [1]. Software architecture research has contributed with the identification of quality attributes (QAs) important for ML-based systems [2], [3] and preliminary catalogs of architectural patterns that address these QAs [4]–[6]. Despite these efforts, it is still very challenging to operationalize ML-based systems as a series of well-understood and balanced architectural design decisions (ADDs) [1].

A common challenge for practitioners is to resolve the performance degradation of ML models, often caused by changes in the data distribution or errors in the ML pipeline [7]. The former issue, commonly referred to as *concept drift* (CD), is of particular concern to practitioners because it can severely affect the quality of service, is difficult to detect, and its resolution can be costly because it often requires infrastructure for monitoring, evolving, and redeploying ML models [8].

Recent research has emphasized the problem of CD in operationalized ML models, with a growing consensus that *observability* is an essential quality of ML-based systems [2], [9]. In addition to observability, practitioners are concerned with *when* to evolve – i.e., retrain or update – their models, and with *how* to evolve them, e.g., what data to include in evolving their models [10]. We refer to this quality concern as the *evolvability* of ML-based systems. Despite the insights of recent research in architectural patterns and tactics for ML-based systems [4]–[6], limited knowledge is still available on the ADDs specifically targeting evolvability. Given its prominence and importance, we believe that the design decisions regarding evolvability and their rationale should be better understood and documented. This will improve stakeholder communication, enable the dissemination of best practices in this emerging field and, ultimately, make ML-based systems more reliable.

Although evolvability has only recently started to emerge as a concern in industry, the field of data mining has been studying the problem of CD for more than two decades, which resulted in a wide variety of methods for evolving ML models [11]. These methods can be understood and categorized according to (i) their applicability in different cases of CD, and (ii) their trade-offs between evolvability and other qualities (e.g., resource utilization or inference latency or explainability). Consequently, addressing evolvability in an ML-based system requires knowledge about the application domain and a holistic view of the system and its requirements.

In this paper, we envision a scenario-based framework to support, justify, and underpin the rationale for making ADDs that address the evolvability of ML-based systems.

The remaining of this paper is organized as follows. Section II introduces our framework and describes its elements. In Section III we instantiate our framework in an illustrative case study. The impact and feasibility of our new idea for documenting the ADD rationale to address evolvability is discussed in Section IV. Future plans to further evaluate and refine our framework are outlined in Section V.

II. Architectural Design Decision Framework to Address Evolvability

Our framework consists of a number of concepts and their relations, depicted in Fig. 1. We describe each of them in turn.

A. Artefact

The artefact is the (sub-)system that includes the ML model (usually part of a larger software system) and to which the evolvability concern applies, e.g., a marketing service that

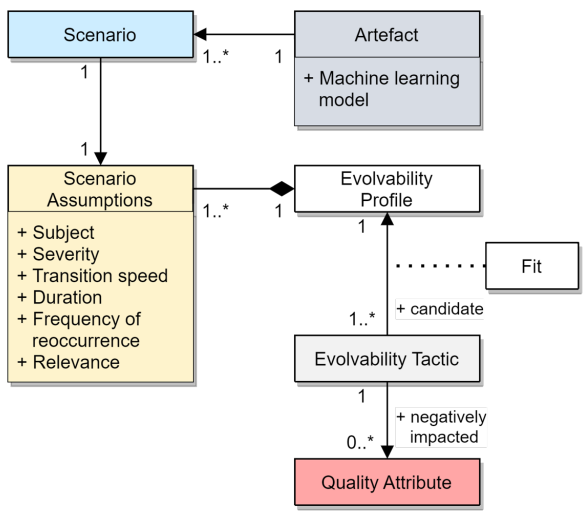

Fig. 1. Architectural design decision framework to address evolvability.

uses a churn prediction model or a photo editing service that uses an object detection model. The utility of the artefact depends on the underlying ML model. For instance, the photo editing service's utility related to providing value-adding functionalities to its end users depends on the accuracy of the used object detection model. It is important to consider the artefact because it provides the context within which evolvability-related scenarios are determined and tactics are eventually evaluated. This essentially also allows us to separate the applicability of evolvability concerns for different subsystems, which is important because multiple ML models in a single system are likely to be subjected to different degrees of non-stationarity. For example, a change in customer demographics may cause CD in a recommendation service, while the autoscaling system remains unaffected as its model does not utilize customer data.

B. Scenarios

A scenario describes a potential instance of CD occurrence in the ML model of the artefact. The identification of scenarios needs to happen by stakeholders that have in-depth knowledge of the artefact's domain in order to be able to foresee changes in the underlying processes (e.g., faults in the data sensing or increase in the load of incoming requests). Such changes may result in important variations of the input data streams of the ML models or the way the outputs are associated with the inputs, or, put simply, in occurrence of CD. The scenarios in our framework have a similar role to QA scenarios for the identification of architecturally significant requirements [12]. The domain knowledge used in the identification of scenarios is vital for the application of our framework.

C. Scenario Assumptions

Once the scenarios are collected, the characteristics of CD [13] for each of them need to be determined and summarized as assumptions using the following template:

- *Subject*. Virtual Drift or Real Drift. The former is a shift in the distribution of the input data $P(x)$; the latter is a shift in the conditional probability distribution of the target variable for a given input $P(y|x)$. Out of the two, real drift is considered more problematic since it always leads to performance degradation [11].
- *Severity*. The degree of change in the data distribution between the time before and after the change.
- *Transition speed*. The time from the start of a shift in the distribution until the distribution becomes stable again.
- *Duration*. The time between the occurrence of two consecutive shifts in the distribution.
- *Frequency of reoccurrence*. How often a certain event that causes CD occurs again.
- *Relevance*. A combination of likelihood and frequency of occurrence of the scenario relative to the other scenarios.

It is possible to deduce certain characteristics directly from the scenario description. For example, a change in the demographic makeup of customers results in virtual drift within a recommendation model that utilizes demographic information as input. Conversely, changes in competitive pressure may result in real drift within a churn prediction model if this is not adequately represented within the model. However, other characteristics such as duration, severity, and reocurrence are less easily deduced at design time and therefore necessitate the formulation of informed assumptions with regard to these characteristics. In our framework, Severity, Transition speed, Duration, Frequency of Reoccurrence, and Relevance are classified on a three-level ordinal scale (low, medium, or high).

D. Evolvability Profile

Effective design decision rationale ultimately improves decision quality and communication, for which it is important to have a structured representation of the arguments [14]. To provide this structure with respect to the assumptions made about the elicited scenarios, we aggregate these assumptions into a single profile that provides a visual representation of the evolvability concern (Fig. 2a). Here, we project the ordinal characteristics of CD to separate dimensions and represent the relevance of the scenarios based on the line width. For the Subject, two separate diagrams need to be created (one for virtual and one for real drift).

E. Evolvability Tactics

An evolvability tactic is an ADD to deal with CD. Given an evolvability profile, there may be several candidate tactics that can address it, but their effectiveness may vary depending on the characteristics of the CD scenario. Our framework represents these characteristics as separate dimensions using the plot of Fig. 2b, where the effectiveness of a tactic in each dimension and scale is represented as white (ineffective), light grey (partially effective), or dark grey (fully effective). This representation corresponds to the architectural knowledge of evolvability tactics included in the cataloguing process.

Below we list important tactics extracted from the data mining literature.

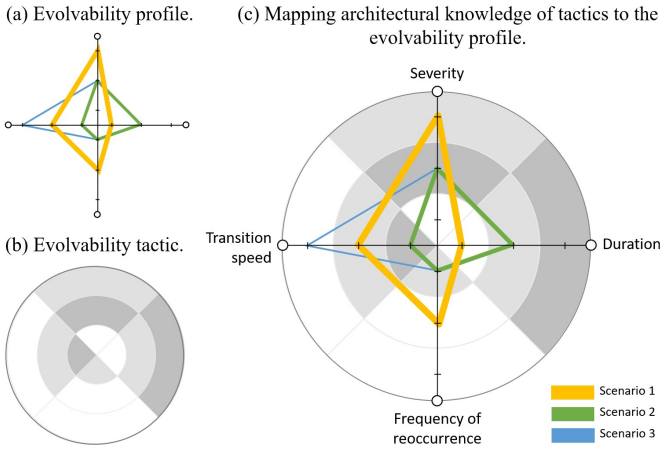

Fig. 2. Representation of the evolvability profile, tactics and mapping. Dimensions have three scales (low, medium, high), starting from the inner quadrant.

1) Online learning: Online learning is an evolvability tactic that differs from the traditional batch training paradigm – training models using the entire training set at once, after which the model remains static – which allows for continuous training of a single model with new observations [11]. Therefore, online learning automatically adapts to CD by effectively weighing the relevance of observations by order of arrival. However, while it is sufficient in evolving the model during gradual and non-severe CD, adaptation to severe and abrupt occurrences of CD is considered slow [15]. Furthermore, the continuous training can cause the model to forget knowledge acquired in previous contexts, also known as catastrophic forgetting [16].

2) Sliding window: Sliding window is an evolvability tactic commonly applied in practice where the model is periodically (e.g. daily or weekly) retrained from scratch using the latest observations [7], [11]. In doing so, the model explicitly forgets knowledge from previous contexts by discarding the oldest data samples from the training window. A sliding window tactic can sufficiently adapt to severe CD and fast transitions between contexts, but is less effective for slow transitions and cases of reoccurring CD [11]. Moreover, determining the size of the window can be difficult, as a small window adapts faster to new contexts while a larger window can provide better performance when the context is stationary.

3) Detection-based model reconstruction: Detection-based model reconstruction is an evolvability tactic in which a new model is reconstructed, either through online learning or batch learning, after detection of CD [11]. In doing so, this tactic can be very effective in adapting to severe and abrupt cases of CD when the model is reconstructed at the right moment [17]. Despite the rather drastic adaptation strategy, which makes it less applicable for short-lasting and non-severe CD. Model reconstruction has been used a lot because it is generally effective in cases where a rapid response to severe CD is required [11], [15]. A wide variety of algorithms have been developed for CD detection, the challenge of which is to identify CD as close to the transition point as possible while remaining robust against false alarms [18].

4) Detection-based model repository: Detection-based model repository is an evolvability tactic that specifically targets reoccurring CD by storing models trained in previous contexts to retain knowledge [19]–[21]. In general, the application of this tactic involves training a model incrementally until drift is detected and then evaluate whether a previous model is suitable for the new context. A detection-based model repository tactic is thus similar to the model reconstruction tactic, but in addition can handle reoccurring CD very well.

5) Adaptive ensemble: An adaptive ensemble tactic aggregates the predictions of multiple models to adapt to CD, typically by training models on different slices of the data stream and dynamically weighting the contribution of each model based on recent prediction performance [15]. These models can be trained either periodically or based on detection, via online learning or sliding windows. Thereby, the adaptive ensemble can be viewed as an extension of aforementioned evolvability tactics. However, it is a more general approach that can adequately handle cases of gradual, abrupt, and reoccurring CD [22], but is potentially less effective compared to approaches that specifically target a particular type of CD (e.g. a model repository for reoccurring CD).

F. Fitting Tactics to the Evolvability Profile

The architectural knowledge of tactics is mapped to the evolvability profile using the representation depicted in Fig. 2c. This representation provides an effective means of compressing domain knowledge about the evolvability scenarios and architectural knowledge of corresponding tactics into a single overview. Based on the overview, the degree of fit for an evolvability tactic can be determined by the number of scenario nodes that are covered by the shaded area in the graph, taking into account the weight of the nodes and shading level of the areas. This degree of fit can then be determined for multiple candidate tactics and, together with the visual overview, be included in the documentation of the rationale.

G. Negative Impact on Quality Attributes

The tactics differ not only in terms of expected fit but also in architectural assumptions regarding system-level components and their interactions, which may adversely affect system QAs. Below we list these assumptions and potentially affected QAs that should be taken into account when evaluating trade-offs.

1) Online architecture: In online learning, the ML model is updated incrementally with new observations from a data stream, as opposed to batch learning where models are trained on a fixed dataset. However, ML models are often deployed as stateless prediction services in elastic service-oriented and microservices architectures [23]. Thereby, the continuous updating of the model's parameters in online learning effectively makes the deployed model stateful [24]. This can adversely affect the *elasticity* of the ML-based system.

2) Batch architecture: Evolvability tactics that train models using batch learning, such as the sliding window tactic, negatively affect the systems qualities in several ways. First, the *training latency* of the system is potentially affected because

the time needed to retrain a model through batch learning can be significantly higher than that of online learning [11], [25]. Second, retraining through batch learning can affect the system's *resource utilization* because of the computational inefficiency of rebuilding the model from scratch at every evolution step and the requirement of storing all the training data into memory [11]. These overheads should be carefully considered especially in resource-constrained domains, such as IoT [26]. Furthermore, the need to store previously acquired samples may also compromise the system's *data privacy* in privacy-sensitive domains, such as healthcare.

3) Ensemble inference: Most methods that deal with CD are model agnostic and thus generally do not interfere with the desired model properties. However, the application of an adaptive ensemble may adversely affect these properties by altering the inference mechanism of the underlying model. As such, this tactic may affect the *explainability* of the ML model, as it can make the model a black-box that is difficult to analyze [27]. Furthermore, the *inference latency* can also be negatively affected because the number of predictions for a single request is effectively multiplied by the number of models maintained in the ensemble [28].

4) Model repository: The system's *resource utilization* can also be negatively affected when a model repository is used to store multiple models, due to the memory overhead [21]. However, the strain on resources depends on the size of the ML model. For example, it may be negligible in applications of decision trees with limited depth, while it can become problematic for neural networks with millions of parameters.

5) Drift detector: The *reliability* of the system can be jeopardized when evolvability tactics are applied that rely on the detection of CD. In production settings, this component is part of the monitoring system of the automated ML workflow [8]. Currently, it remains difficult to detect CD accurately and therefore false alarms are inevitable with automated drift detection mechanisms [29]. Furthermore, CD can also be detected correctly, but only last for a short period of time. Thus, any automatic response to a drift detector requires caution with methods that involve a drastic operation, such as taking a potentially viable model out of deployment.

III. Example of Framework Application

In this section, we instantiate our framework in a churn prediction case for a marketing service to illustrate its application.

During the development of a new subscription-based Web service, stakeholders establish a business objective to minimize customer churn. A core functionality of the system is a marketing service that offers customers personalized discounts based on the predicted likelihood for a customer to churn. In the elicitation process, evolvability is raised as an important concern because the business operates in a growing and highly competitive market and, therefore, volatile patterns are expected in the context of this service that could potentially deteriorate the performance of the underlying ML model.

Fig. 3 depicts the instantiation of our ADD rationale framework for the illustrative case. We can consider that three

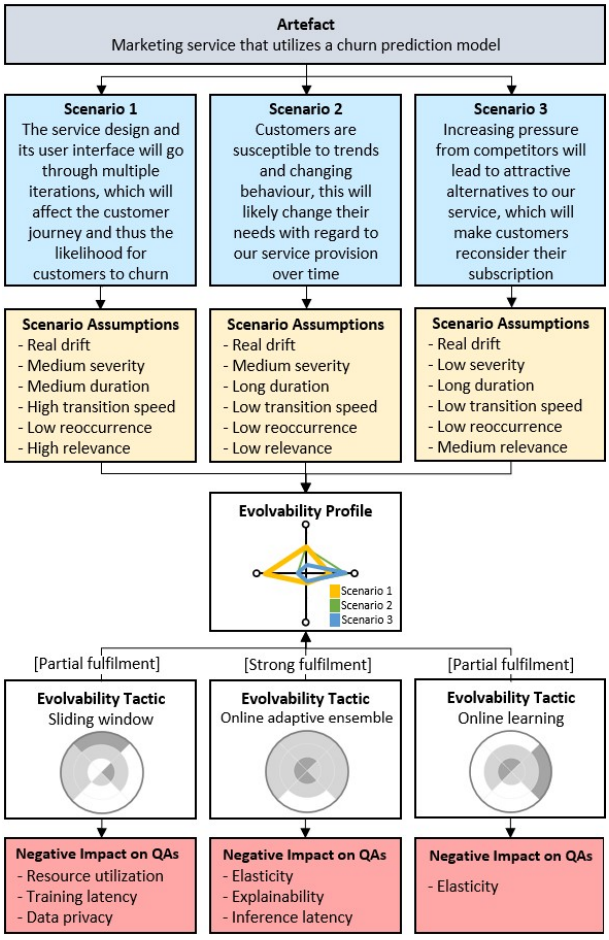

Fig. 3. Framework instantiation for the churn prediction model.

scenarios are elicited and projected in an evolvability profile, for which two evolvability tactics are identified that partially fulfill the profile and one that strongly fulfills it. Despite the partial fulfilment of the sliding window tactic, it can be chosen in favour of the other candidate tactics if a negative impact on elasticity needs to be avoided.

IV. Discussion

Our framework establishes the design decision process to explicitly and systematically address the evolvability of ML models, starting from the expectation of CD in the data distribution and concerns about performance degradation.

A key requirement for our approach is a catalogue of architectural tactics for evolvability that encodes the architectural knowledge regarding the efficacy of tactics on different CD cases and their negative impacts on other QAs. Limited knowledge is currently available in this regard. In this paper, we derived a preliminary catalogue of tactics from the data mining literature; this initial set needs to be refined and extended by deriving tactics from industrial practice and experience. Furthermore, the evolvability profile, in conjunction with its corresponding mapping to architectural knowledge of evolvability tactics, constitutes a fundamental aspect of our

framework in determining the suitability of potential tactics. The visual representation of this profile that we propose has the potential to serve as a valuable tool for facilitating communication with stakeholders and making informed trade-offs. An interesting direction for future research is to extend the framework to include the derivation of the expected response measure of candidate tactics for a given evolvability profile, thereby providing additional support in evaluating trade-offs.

In addition to evolvability, we argue that the framework can be extended in a straightforward way to address the observability of ML-based systems, since CD is also relevant to this important QA [2]. At the same time, we believe a similar approach to ours can be applied to other key QAs of ML-based systems, such as explainability and trust. All this contributes to more and better architectural support for the emerging discipline of ML operations (MLOps) [7], [8].

V. Future plans

The main idea behind our proposal in this paper is that the course of CD can be predicted and addressed based on domain knowledge. Starting from this initial hypothesis, we plan to:

(1) Apply our framework in a number of case studies to (a) evaluate its applicability in practice, (b) extract and catalogue more evolvability tactics that can be reused within an application domain. We intend to do this in the frame of ExtremeXP, a new EU project focusing on flexible data analytics and ML [30].

(2) Gather empirical data on the effectiveness of the framework in guiding ADDs and its embedding in everyday architecting practices via a series of interviews with industry experts from different application domains.

(3) Evolve the framework incrementally by the iteratively performing steps (1) and (2).

(4) Investigate how a system can continually refine the evolvability profile and switch between evolvability tactics at runtime based on their cost and benefit [31], [32].

Acknowledgment

This research is supported by ExtremeXP, a project co-funded by the European Union Horizon Programme under Grant Agreement No. 101093164.

References

[1] A. Serban and J. Visser, "An empirical study of software architecture for machine learning," *CoRR*, vol. abs/2105.12422, 2021. [Online]. Available: https://arxiv.org/abs/2105.12422

[2] G. A. Lewis, I. Ozkaya, and X. Xu, "Software architecture challenges for ML systems," in *ICSME*, 2021, pp. 634–638.

[3] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, and S. Wagner, "Software engineering for ai-based systems: a survey," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–59, 2022.

[4] H. Washizaki, F. Khomh, Y.-G. Guéhéneuc, H. Takeuchi, N. Natori, T. Doi, and S. Okuda, "Machine learning architecture and design patterns," *IEEE Computer*, vol. 55, 2022.

[5] S. K. Lo, Q. Lu, L. Zhu, H.-y. Paik, X. Xu, and C. Wang, "Architectural patterns for the design of federated learning systems," *Journal of Systems and Software*, vol. 191, p. 111357, 2022.

[6] L. Myllyaho, M. Raatikainen, T. Männistö, J. K. Nurminen, and T. Mikkonen, "On misbehaviour and fault tolerance in machine learning systems," *Journal of Systems and Software*, vol. 183, p. 111096, 2022.

[7] S. Shankar, R. Garcia, J. M. Hellerstein, and A. G. Parameswaran, "Operationalizing machine learning: An interview study," *arXiv preprint arXiv:2209.09125*, 2022.

[8] D. Kreuzberger, N. Kühl, and S. Hirschl, "Machine learning operations (mlops): Overview, definition, and architecture," *arXiv preprint arXiv:2205.02302*, 2022.

[9] S. Shankar and A. Parameswaran, "Towards observability for machine learning pipelines," *arXiv preprint arXiv:2108.13557*, 2021.

[10] K. M. Habibullah and J. Horkoff, "Non-functional requirements for machine learning: Understanding current use and challenges in industry," in *Requirements Engineering*, 2023.

[11] J. a. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, 2014.

[12] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 4th ed. Addison-Wesley Professional, 2021.

[13] G. I. Webb, R. Hyde, H. Cao, H. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Min. Knowl. Discov.*, vol. 30, no. 4, 2016.

[14] D. Falessi, M. Becker, and G. Cantone, "Design decision rationale: experiences and steps ahead towards systematic use," *ACM SIGSOFT Softw. Eng. Notes*, vol. 31, no. 5, 2006.

[15] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2019.

[16] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[17] A. Tahmasbi, E. Jothimurugesan, S. Tirthapura, and P. B. Gibbons, "Driftsurf: A risk-competitive learning algorithm under concept drift," *arXiv preprint arXiv:2003.06508*, 2020.

[18] A. Pesaranghader, H. L. Viktor, and E. Paquet, "Mcdiarmid drift detection methods for evolving data streams," in *Intl Joint Conference on Neural Networks*, 2018, pp. 1–9.

[19] J. Gama and P. Kosina, "Tracking recurring concepts with meta-learners," in *Portuguese Conference on Artificial Intelligence*, ser. LNCS, vol. 5816. Springer, 2009, pp. 423–434.

[20] A. M. Ángel, G. J. Bartolo, and M. Ernestina, "Predicting recurring concepts on data-streams by means of a meta-model and a fuzzy similarity function," *Exp Sys with App*, vol. 46, pp. 87–105, 2016.

[21] R. Anderson, Y. S. Koh, and G. Dobbie, "Cpf: Concept profiling framework for recurring drifts in data streams," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 203–214.

[22] M. Han, Z. Chen, M. Li, H. Wu, and X. Zhang, "A survey of active and passive concept drift handling methods," *Comput. Intelligence*, 2022.

[23] M. Hamilton, N. Gonsalves, C. Lee, A. Raman, B. Walsh, S. Prasad, D. Banda, L. Zhang, L. Zhang, and W. T. Freeman, "Large-scale intelligent microservices," in *Big Data*. IEEE, 2020, pp. 298–309.

[24] S. Zhao, X. Chen, C. Wang, F. Li, Q. ping Ji, H. Cui, C. Li, and S. Wang, "Hams: High availability for distributed machine learning service graphs," *DSN*, pp. 184–196, 2020.

[25] T. Chen, "All versus one: an empirical comparison on retrained and incremental machine learning for modeling performance of adaptable software," in *SEAMS*, 2019, pp. 157–168.

[26] H. Mehmood, P. Kostakos, M. Cortes, T. Anagnostopoulos, S. Pirttikangas, and E. Gilman, "Concept drift adaptation techniques in distributed environment for real-world data streams," *Smart Cities*, vol. 4, 2021.

[27] D. Sepiolo and A. Ligeza, "Towards explainability of tree-based ensemble models. a critical overview," in *International Conference on Dependability and Complex Systems*. Springer, 2022, pp. 287–296.

[28] K. Sarpatwar, K. Nandakumar, N. Ratha, J. Rayfield, K. Shanmugam, S. Pankanti, and R. Vaculin, "Efficient encrypted inference on ensembles of decision trees," *arXiv preprint arXiv:2103.03411*, 2021.

[29] G. A. Lewis, S. Echeverría, L. Pons, and J. Chrabaszcz, "Augur: A step towards realistic drift detection in production ml systems," in *SE4RAI*, 2022, pp. 37–44.

[30] "EXPeriment driven and user eXPerience oriented analytics for eXtremely Precise outcomes and decisions Horizon project," https://tinyurl.com/25nym7k5, expected to start in Jan 2023.

[31] I. Gerostathopoulos, C. Raibulet, and E. Alberts, "Assessing self-adaptation strategies using cost-benefit analysis," in *ICSA 2022 Companion*, 2022, pp. 92–95.

[32] M. Casimiro, P. Romano, D. Garlan, G. A. Moreno, E. Kang, and M. Klein, "Self-adaptation for machine learning based systems," in *ECSA (Companion)*, 2021.