J Grid Computing (2024) 22:71
https://doi.org/10.1007/s10723-024-09785-z

RESEARCH

Adapting to Changes: A Novel Framework for Continual Machine Learning in Industrial Applications

Jibinraj Antony · Dorotea Jalušić · Simon Bergweiler · Ákos Hajnal · Veronika Žlabravec · Márk Emődi · Dejan Strbad · Tatjana Legler · Attila Csaba Marosi

Received: 13 May 2024 / Accepted: 22 October 2024

Abstract This paper is dedicated to solving the problem of concept drift in industrial plants using artificial intelligence methods. For this purpose, methodological approaches and procedures are considered and analyzed. Based on the findings, reference architectures were developed at different abstraction levels that can be used in an industrial environment and enable continuous machine learning. Continuous machine learning offers the possibility of adapting to dynamic changes in the production environment, which are reflected in constantly changing data sets. Through a combination of machine learning techniques, a novel and practical framework for continuous learning, also known as lifelong learning, is presented. The integration of problem-focused machine learning methods is advancing in production, e.g., predictive maintenance, process optimization or fault detection. Thereby, fully or semi-automated adaptations to changing environments requiring continuous improvements are less often explored, although practical use cases often require adaptive capabilities as the physical data distribution may change over time. In this paper, the application was continuously improved based on case studies and empirical results, and finally validated with a quality assurance application. Various methods and approaches for detecting concept and data deviations, retraining, packaging and model updating had to be

Jibinraj Antony, Dorotea Jalušić, Simon Bergweiler, Ákos Hajnal, Veronika Žlabravec, Márk Emődi, Dejan Strbad, Tatjana Legler, and Attila Csaba Marosi contributed equally to this work.

J. Antony (✉) · S. Bergweiler · T. Legler
German Research Center for Artificial Intelligence (DFKI), Trippstadter Str. 122, 67663 Kaiserslautern, Germany
e-mail: jibinrajantony@gmail.com

S. Bergweiler
e-mail: simon.bergweiler@dfki.de

T. Legler
e-mail: tatjana.legler@dfki.de

D. Jalušić · V. Žlabravec · D. Strbad
Ascalia Ltd., Trate 16, 40000 Čakovec, Croatia
e-mail: dorotea.jalusic@gmail.com

V. Žlabravec
e-mail: veronika@ascalia.io

D. Strbad
e-mail: dejan@ascalia.io

Á. Hajnal · M. Emődi · A. C. Marosi
Laboratory of Parallel and Distributed Systems, Institute for Computer Science and Control, Hungarian Research Network, Budapest, Hungary
e-mail: akos.hajnal@sztaki.hun-ren.hu

M. Emődi
e-mail: mark.emodi@sztaki.hun-ren.hu

A. C. Marosi
e-mail: attila.marosi@sztaki.hun-ren.hu

Á. Hajnal · M. Emődi
John Von Neumann Faculty of Informatics, Obuda University, Budapest, Hungary

Published online: 29 November 2024

investigated, which led to the question of what a real industry-oriented implementation could look like. The result is a reference architecture that can run on cloud and edge computing resources. This reference architecture is validated in real-world application in the parquet production sector, proving its feasibility and efficiency.

Keywords Continual machine learning · Continuous machine learning · Concept drift · Automated deployment · Reference architecture · Manufacturing-as-a-service

1 Introduction

Machine Learning (ML) has revolutionized the manufacturing industry by enabling manufacturers to increase efficiency [1, 2], reduce costs [3, 4] and improve quality [5, 6]. ML has proven its applications in many fields, such as predictive maintenance [7, 8], quality control 9, 10, process optimization [11, 12], etc.

In a conventional ML application, the ML Algorithm identifies the features or patterns in a historical dataset called *training dataset* and adapts the ML Model parameters according to the identified features during the Model *training* or *learning* process. The given training dataset shall have a static nature and be a subset of the data points from the real-world application, which have been collected historically over time and have similar features to those of the application. After successfully completing the model training process, the trained ML model will be able to imitate the given data distribution through its parameters (weights) and make predictions on any new given data point with features similar to the training data distribution. This data-driven nature of ML Algorithms makes them highly dependent on the quality and quantity of the collected historical data points and their effectiveness in representing the real application scenario, which in turn becomes a crucial factor in the success of the application use case.

In a dynamically changing world, where the nature and features of products continuously vary, the static nature of the dataset used in ML applications creates inefficiencies and uncertainties in the long run. This idea of *Concept Drift*, where the underlying data distribution changes over time, thereby making the trained ML models obsolete, has become a critical issue in the trustworthiness of ML models in a production environment. For example, an ML model used to predict frauds in credit card transactions may become less accurate as fraudsters change their tactics [13]. To dynamically adapt to the changes in the real world and to eliminate the *Data Drift* inefficiencies, modern ML techniques such as Continual Learning (CL) and implementation approaches such as Continuous Machine Learning (CML) must be used.

Continual Learning, also referred to as *lifelong, sequential, or incremental learning,* is the problem of learning from an infinite stream of data with the intent to preserve and extend the acquired knowledge [14]. In contrast to static artificial neural networks, which are incapable of adapting or expanding, real-world streams of information constantly evolve [14]. Humans learn sequentially throughout their lives. They gradually forget some old information, especially details, but without completely losing obtained knowledge [15]. The ability to integrate new knowledge and stability to preserve previous knowledge is called the "stability-plasticity" dilemma in both biological [16] and artificial neural systems [17]. Without a good balance, neural networks can suffer from "catastrophic forgetting", thus resulting in a deterioration of performance in handling old classes when acquiring new data [18]. Although continual learning is a long-standing challenge [19, 20], with the spread of AI applications in the daily life, the ability to continuously learn is still of high importance.

On the other hand, Continuous Machine Learning is the latest ML development strategy for production-ready ML applications, where an ML Model has been integrated and managed at periodic instances, leveraging *Agile DevOps* practices used in the industry. Although such a continuous approach can address a few requirements of a dynamic real world, it still associates other challenges such as complex infrastructure requirements, low failure rates, etc. This gap can be filled by combining the approaches of CML with CL tools to achieve a highly dynamic ML workflow, which can be efficiently transferred to industrial scenarios.

Machine Learning Operations (MLOps) [72] refers to the efficient lifecycle management of machine learning models, including the technologies and methodologies required. It is considered a multidisciplinary field, combining the practices of Machine

Learning, DevOps (Development and Operations), and data engineering. The main goal of MLOps is to improve the efficiency, quality, and maintainability of machine learning projects by automating and standardizing the process of taking a machine learning model from development to production and managing its complete lifecycle. MLOps practices can be applied to any machine learning project and itself does not concern with the learning methods used within a project. The Continual Machine Learning Reference Architecture presented in this paper incorporates many MLOps aspects in an automated manner, such as re-training, versioning, testing, deployment, and monitoring and management.

DIGITbrain [21] is a European innovation project that aims to support the manufacturing industry by providing a collaborative platform for faster development of Digital Twin (DT) solutions, re-usability and dynamic composition of various assets [22] (*models* incorporating behavioral, physics, finite element model, 3D, machine learning [23], dynamically bound *data resources*, and *algorithms* decomposed into *microservices*), and easier access to high-performance computing and cloud resources.

Reference Architectures (RAs) represent frequently emerging patterns and best practices, which can be defined at various levels of abstraction. For example, "Reference Architectural Model Industry 4.0" [24] (RAMI 4.0) or "Industrial Internet Reference Architecture" [25] is built on high-level concepts. At the same time, RAs provided by popular public cloud providers such as Amazon WebServices (AWS) [26], Google Cloud Platform, and Microsoft Azure [27] are more concrete, directly deployable components. RAMI 4.0 represents a high-level (abstract) reference architecture, a framework for digital transformation in manufacturing and industry that facilitates the integration of digital and physical processes. While being comprehensive, it is not without difficulties, such as (i) the cost of implementation to align with the standards of RAMI 4.0; (ii) complexity as it is a multi-dimensional and multi-layered model; (iii) it has its adaptability, flexibility and interoperability issues; and (iv) assumes organizational and process changes within any organization to adapt the model. Generally, it provides a structured approach for companies to modernize and standardize their operations. Still, it may not be suited for organizations of all sizes and maturities and may not be applied to improve only a specific set of processes. DIGITbrain specializes in providing smaller-scale best-practice recommendations, blueprints, patterns, and prototypes in manufacturing context [28–31], which can be directly applied or customized with little effort. Research and development carried out were validated using real-life industrial use cases.

This paper focuses on conceptualizing an industry-transferable, dynamic ML pipeline and validating it in a Quality Control operation use case, namely, in the glue application process at Bauwerk Group Hrvatska d.o.o. *Reference Architectures* are provided for dynamic ML at different levels of abstraction (from schematical to concrete realization). The process and the presented results can be utilized in other industrial use cases or at least can serve as a starting point or template for similar applications.

The main contributions of this work are as follows:

- overview and analysis of related approaches and methodological solutions addressing the problem of concept drift in industrial applications
- a logical architecture enabling continual machine learning
- a concrete reference architecture realization applicable in industrial contexts
- a case study and empirical results that validate the approach to an industrial quality control application

Note that the paper does not merely aim to provide a realization of a particular continuous machine learning application, which though can be applied in various domains in the industry, but to draw the attention to the phenomenon of concept drift, which can explain why the effectiveness of the applied machine learning technique degrades over time. We present an overview of the state-of-the-art methodologies and concepts in the field of continual learning. The proposed high-level, logical reference architecture can serve as a functional blueprint, a starting point for designing such a complex system, and we also present a particular implementation option, which was validated on a real-world manufacturing process to show its applicability and benefits for manufacturing companies. To our experiences the introduction of ML is ongoing and of very high interest at small and medium enterprises (SMEs) having huge potential and even higher expectations. ML technologies

are more mature and are better supported by several tools and research materials. Application of continual machine learning however is less known, still a very important from both practical and research points of view.

The rest of the paper is organized as follows: In the next section, we overview the problem of concept drift in static ML models, and related, existing methodological approaches addressing it. We then propose both a logical and a specific Reference Architecture that can help realize Continual ML architectures and workflows. It is followed by a particular use case and empirical results on adapting the proposed Reference Architecture onto an industrial application. Finally, the paper is concluded.

2 Related Work

Although ML utilizes static, historical data for its development, it has been successfully applied to diverse use cases in various branches of industry: agricultural industries for the optimization of food production [32], medical applications for disease diagnosing [33], making cybersecurity systems more secure [34], Industry 4.0 applications [35] etc.

The dynamic characteristics of industrial use cases and the inability of ML models to adapt to changes, as addressed in the context of *concept drift* or *drift* [36], make ML models obsolete in certain real-world environments and applications. ML models which are incapable of detecting cyber attacks as the attackers change their strategies [13] or the medical image processing models which deteriorates its performance as the diversity of the data increases [73] are some of the examples of such real-world applications. On the other hand, the relationship between input and output data can also change over time meaning that, in turn, there are changes to the underlying mapping function *learned* by the ML model. These changes may be consequential, such as the predictions made by a model (trained on historical data) are no longer correct or as correct as they could be if the model would have been trained on more recent historical data. Such unforeseeable changes of the underlying data distributions result in poor learning results [37] and performance deterioration [38] over a certain period. For successful ML applications, it is thus essential to take *drift* into consideration even during the development phase to recognize the drift in early stages and to deploy updated models that would diagnose themselves and adapt to changes in data over time [39], which are essential in error-free industrial applications.

Early detection of drift, characterization and quantification, identification of change points and change time intervals [40] are important topics in many ML applications. Generally, drift detection algorithms can be classified into three different categories [37]. In an *error rate-based* drift detection algorithm, the error rate of the base classifier is tracked, and a drift notification will be triggered based on a statistically significant error rate. Common algorithms such as Drift Detection Method (DDM) [41], Learning with Local Drift Detection (LLDD) [42], Early Drift Detection Method (EDDM) [43] use an error rate-based approach for drift detection. In *data distribution-based* drift detection approaches, algorithms use a metric or distance function to quantify the dissimilarity between the distribution of historical and new data points, and at statistically significant dissimilarity instances, a trigger has been initiated. In such data distribution-based drift detection methods, the algorithms consider the distribution drift to quantify the drift. In the third category, called *multiple hypothesis test* drift detection [37], a combination of both the *error rate* and the *data distribution* approaches has been utilized. Although these drift detection algorithms can identify the occurrence time (when) and the severity of the concept drift (how), only a very few algorithms can locate the drift regions (where) in the dataset [37].

Upon successful detection of drift, a general strategy for *drift adaptation* or *drift reaction* is to perform a *simple model retraining*, develop *ensemble models,* or perform *model adjusting* to handle different types of drift [37]. The retraining strategy is one of the most successful methods of addressing global drift by replacing the obsolete model with a new one. The DELM [44] model performs such a retraining approach to address the global drift by actively adjusting the number of hidden layer nodes for classification models. Similarly, the *just-in-time* approach used in [45, 46] detects the drift using kNN classifiers [46], and the models are updated on-demand.

Model ensemble approaches were found to be more efficient in addressing the recurring concept drifts, where the previously trained models are

reused, and retraining efforts are minimized. In a general ensemble approach, a combined result from different models (using certain voting rules and weighting) is generated and used to make new predictions. The approaches proposed by [47, 48] utilize ensemble models where an algorithm replaces the worse-performing model with the newer model to address the recurring drift.

Concept drift handling in industrial applications is challenging, involving performance as well as safety risks for the processes. In an industrial process, a monitoring application of a CNC Milling use case [49] proposes a data drift tracking method to evaluate the uncertainty of utilized CNN's. Once the drift is detected, new training samples are generated, and the model is adapted to the latest changes by retraining. Similarly, in an Injection Molding process's quality control system [50], various drift detection algorithms are utilized to identify the drift, and the corresponding adaptations are performed again by retraining the model on new training datasets. Although these approaches address the concept drift in the application, dynamic adaptations of data changes in a continuous manner are rarely researched and not widely implemented in industrial applications.

CL has originated as a new ML concept where a model learns from a dynamically changing dataset and updates and exploits its knowledge over its lifetime. Although CL enables models to develop themselves adaptively, it is limited by its *catastrophic forgetting* nature, where learning new tasks results in performance degradation on old tasks. In comparison to the traditional ML approaches, the dynamic adaptability of the CL models enables them to perform better in real-world problems, as explained by the plasticity-stability dilemma, with plasticity referring to the ability to integrate new knowledge and stability retaining previous knowledge while encoding it [51].

Traditional ML neural networks tend to be overly plastic, lacking the stability necessary to prevent forgetting previous knowledge, and as the learning progresses, networks tend to forget previously seen tasks, which have been successfully addressed by various families of CL approaches [52]. Various methods [53–55] use regularization-based techniques while retraining the whole network; methods that selectively train the network and dynamically expand it on-demand to represent the new tasks by architecture modifications [56, 57]; methods using memory replay to consolidate internal representation [58, 59]; methods by explicitly designing and manipulating the optimization programs [60–62] have successfully implemented strategies to overcome catastrophic forgetting [14, 63], achieving a better plasticity-stability trade-off.

Due to the complexity of the real world, the practical application of Continual learning is facing many difficulties, namely, scenario complexity, task specificity and scarcity of labeled data [63], decelerating industrial adaptation of these approaches. Although concept drift is a well-known challenge in ML applications for industrial use cases, the application of Continual learning is not broadly researched and transferred to the industrial use cases.

3 Continual Machine Learning Reference Architecture

This section presents a schematic block diagram, a logical structure of a Continual ML reference architecture, incorporating components necessary to realize potential CML workflows. We describe all the major steps of such processes, starting from the "hot" control loop performing real-time data analysis and intervention, then extending to the "cold" path that allows to detect concept drift to re-train and re-deploy the refined model from time to time to adapt to the changing environment. It is followed by an overview of existing tools and software frameworks that can be facilitated to realize such an end-to-end infrastructure, in part or fully. Finally, we propose a concrete and complete Continual ML Reference Architecture, in which every logical component has its corresponding software choice, and so thus, after an application-specific customization, it can directly be deployed and run in a local or cloud-edge execution environment, respectively.

3.1 High-level CML Reference Architecture

Fig. 1 illustrates the structure of the logical reference architecture composed of high-level functional blocks, which can serve as a blueprint, pattern, or simply a schematic guideline to overview, understand what needs to be designed and implemented to construct a Continual ML application. This

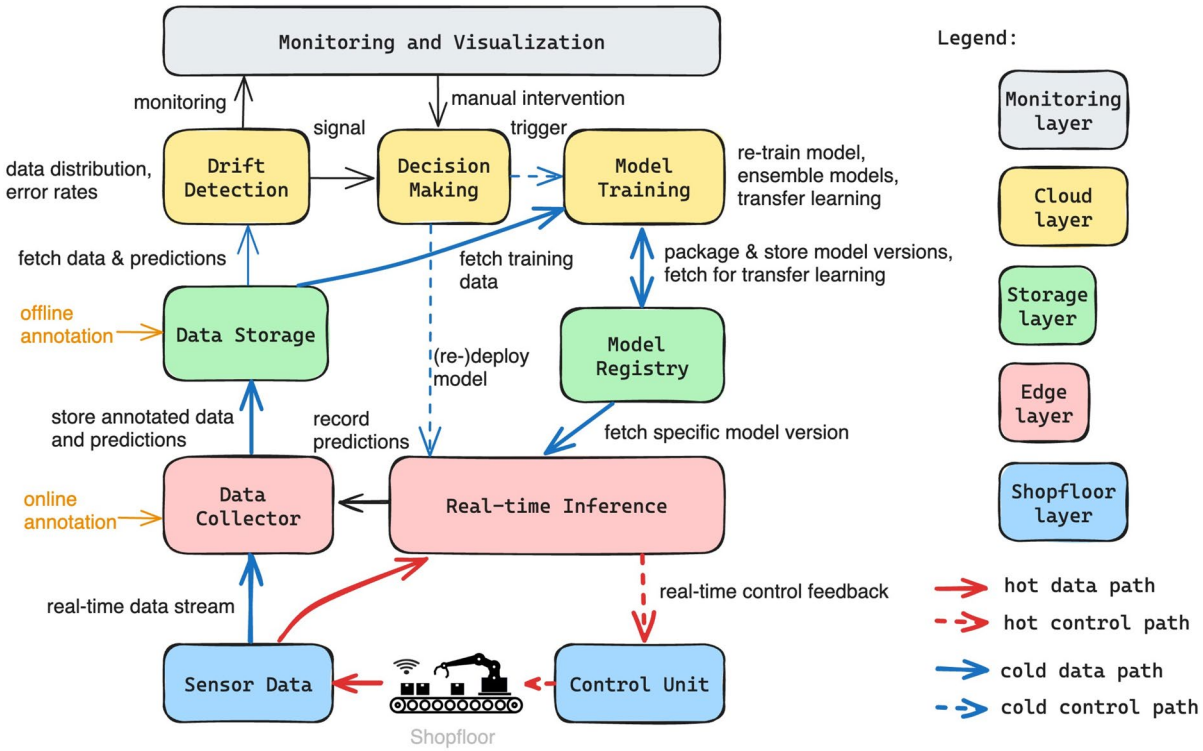

Fig. 1 High-level CML reference architecture and workflows

architecture contains all the essential, necessary modules and functionality, aimed to be as complete as possible, although, note that, in practice, several other components might extend this base architecture to fulfil further application-specific requirements. This model was constructed jointly by several companies from the industry and research organizations specialized for IT and AI technologies, went through multiple iterations of developments, and combines research and industrial practices gained from small and medium enterprises (SMEs) during the DIGITbrain project in manufacturing domain. In the following paragraphs, we explain the role and functionality of the individual components.

Proceeding upwards from the bottom, the lower part represents the shopfloor in a production plant with a production machine equipped with several sensors observing various metrics of the live manufacturing process. Examples of production machines include metal-forming press machines or industrial robot arms, but they can also be complex systems with several cooperating components (even geographically distributed). Examples of sensors include heat-, pressure sensors, industrial cameras, programmable logic controllers (PLCs), or other software components (e.g., Enterprise Resource Systems), databases, respectively.

The stream of sensor data is fed into a component called Real-time Inference. It performs real-time inferences/predictions using ML algorithms and models (initially) trained on historical data previously. The result of the inference can imply immediate control feedback and intervention to the production machine (or another cooperating machine) through a component called a Control Unit. This loop of data and control is called the “hot path” (indicated as red arrows), as it is used actively in the production process.

The data coming from the sensors is forwarded to a Data Collector component, which is responsible for storing new data for later analysis, and might also perform further data transformation and preprocessing activities such as data augmentation,

scaling, sampling, cleaning, aggregation, KPI (Key Performance Indicator) extraction, etc. Data Collector also records the actual predictions made by the current ML application (Real-time Inference block) and potentially collects and assigns annotations (labels) for data samples on the fly, entered by a human operator.

Data Collector can also buffer the data before sending it to a persistent storage system called Data Storage. This direction of data is called a "cold path", which is typically it is less time-critical compared to the "hot" control loop. Data Storage can be an object storage, as an example, a database, or a message broker, respectively. It can be hosted on-premises (keeping governance, control, and sovereignty over the data) but can potentially hosted in a cloud storage service outside the facility.

Drift Detection module continuously analyses new data (obtained from the Data Storage component), derives and calculates statistical, metric data, trends such as accuracy of prediction degradation over time, distribution change and other anomalies.

A Monitoring and Visualization layer, which is typically the graphical user interface of the system, can visualize this information for a human operator (but can provide other monitoring and status information about the system; this time we restrict our attention of CML aspects). When drift is detected, it notifies the Decision Making component, which is responsible for determining whether model re-training is required and when it can be performed (e.g., enough new training data is available, drift exceeds a certain threshold), it schedules re-training and determines the set of data to be used (only new or a mixture of old and new, etc.) and whether a new model architecture is required (e.g., in case of a new category or ensembles). The decision may require user interaction or confirmation.

Model Training module performs the training processes. This component is typically hosted in an execution environment equipped with sufficient compute resources (typically with GPU cards). Model training may use previous model parameters as a starting point (transfer learning), and the resulting model is eventually stored in the Model Registry using versioning and other metadata.

When the new model becomes available and is uploaded into the Model Registry, the Decision Making component initiates graceful termination of the current control process and instructs the Real-time Inference module to fetch the new model version (indicated by "(re-)deploy model" arrow), apply it and resume the manufacturing with as little downtime as possible.

Note that this is a high-level reference architecture that includes functional components needed to cover workflows in CML systems. Although it is also an architecture of high abstraction, in contrast to RAMI 4.0 for industry, this model focuses on CML. It intentionally does not specify software solutions and other implementation details that could restrict the application of the logical scheme to use cases or uniform requirements. It provides merely hints for placement for the components (shopfloor, edge, cloud), which can vary from application to application; an "optimal" placement cannot be determined in general in advance and is largely dependent on company policies and preferences. In extreme cases, it can happen that all components are best deployed on the shopfloor (no data leaves company premises), or everything runs in the cloud (no infrastructure is needed from the company), respectively. In most cases, a mixed allocation of functional components to compute and storage resources might be preferable, considering several aspects like performance, cost, data privacy, etc. Because the necessary data and model storage technologies (and the required infrastructure backing) also vary widely (databases, object/file storages, streaming resources, potentially storing loads of volumes in a distributed and replicated manner), no placement is recommended at all for these components (simply referred to as a storage layer). Re-training, due to its high computing requirements (GPU support) and its periodical but temporary nature, is typically allocated to clouds for cost-performance tradeoff reasons (as recommended by the architecture).

3.2 Related Tools and Frameworks

There are several platforms and libraries available to support continual ML use cases. MLflow [64] is an open-source platform designed to streamline the machine learning life cycle, manage and track machine learning experiments, models, and deployments. It provides a unified interface and allows users to work with various machine-learning frameworks, libraries, and tools. The core components of

MLflow include (i) Tracking, which enables users to log and organize experiments and their parameters, metrics, and artifacts; (ii) Projects, which standardize the structure and modularization of machine learning code to make it more shareable and reproducible; and (iii) Models, which facilitate model packaging and deployment across different platforms.

Additionally, MLflow offers a Model Registry for version control, collaboration on model development and an extensive set of APIs and integrations. Furthermore, MLflow Recipes is an experimental framework for structuring MLOps workflows that simplifies and standardizes machine learning application development and productionization. It provides (a) predefined templates that include standardized stages such as data ingestion, transformation, training, evaluation, etc.; (b) a recipe execution engine that allows fast iteration by, e.g., keeping track of the dependencies within any pipeline and will trigger re-execution starting only from the required step; and (c) standardized and modular structure to simplify and automate hand-off from development to production.

On the other hand, Avalanche [65] is an open-source end-to-end *Continual Learning* library based on *PyTorch*. Its goal is to support fast prototyping, training and evaluation of continual learning algorithms. Its initial focus was on continual supervised learning for vision tasks. It focuses on reproducibility, scalability, and code efficiency (e.g., requiring less code, allowing faster iteration, and reducing errors). It consists of five modules: (i) the benchmarks module contains the major CL benchmarks and offers a uniform API for data handling; (ii) the training module contains functionality for model training, including efficient methods for implementing new CL strategies and also contains CL baselines and algorithms to use and compare against; (iii) the evaluation module contains functionality for evaluating CL algorithms based on the criteria, factors that the Avalanche authors consider important; (iv) the models module includes task-aware models and utilities to expand models through a set of pre-trained models and popular (reference) architectures, similarly to *torchvision.models*. Finally, (v) the logging module contains logging and plotting features, including native support for *Tensorboard*. As Avalanche is an off-the-shelf open-source framework offering end-to-end CL implementation and customization opportunities on demand, its functionalities are in line with the objectives of the paper and, therefore, chosen for the implementation.

3.3 Concrete CML Reference Architecture

Figure 2 illustrates the proposed Reference Architecture implementation. It contains three major layers. (i) The DIGITbrain Platform (also called Digital Agora) serves as a high-level user interface (also, the control and orchestration layer within the ecosystem). (ii) The cloud layer is designed to provide the necessary compute capacity from a cloud. Throughout the deployment process, the platform orchestrates the deployment of the predefined components and facilitates essential integration steps, such as data management and communication between the components; (iii) The infrastructure of the manufacturing company, including the Data Storage and Model Registry components.

In manufacturing, data handling and control feedback are generally sensitive topics; therefore, companies often prefer to share as little information as possible with the outside world (and the competitors). Therefore, the Data Storage and the Control Unit (which beyond privacy has high security risk) are separated from the provided reference architecture implementation and are connected dynamically on application start on-the-fly.

During the process of data integration, data and models need to be stored and accessed, but since the Reference Architecture does not intend to delimit the type of storage to be used (which varies significantly from use case to use case depending on the actual data the factory produces), it contains an Rclone component to connect to these external storages seamlessly (data and model storages may even be of different types). Rclone offers dynamic configuration options to connect to a particular storage at run-time, supporting more than 70 different providers (including AWS S3, Microsoft Azure Blob Storage, Google Drive, etc.). Moreover, leveraging the functionality provided, Rclone component ensures that only essential data is synchronized to the cloud. This targeted synchronization minimizes network load, reduces data movement latency, and carefully considers the secure advantages associated with retaining data on-premises.

For the CL framework Avalanche had been selected as a software implementation. The

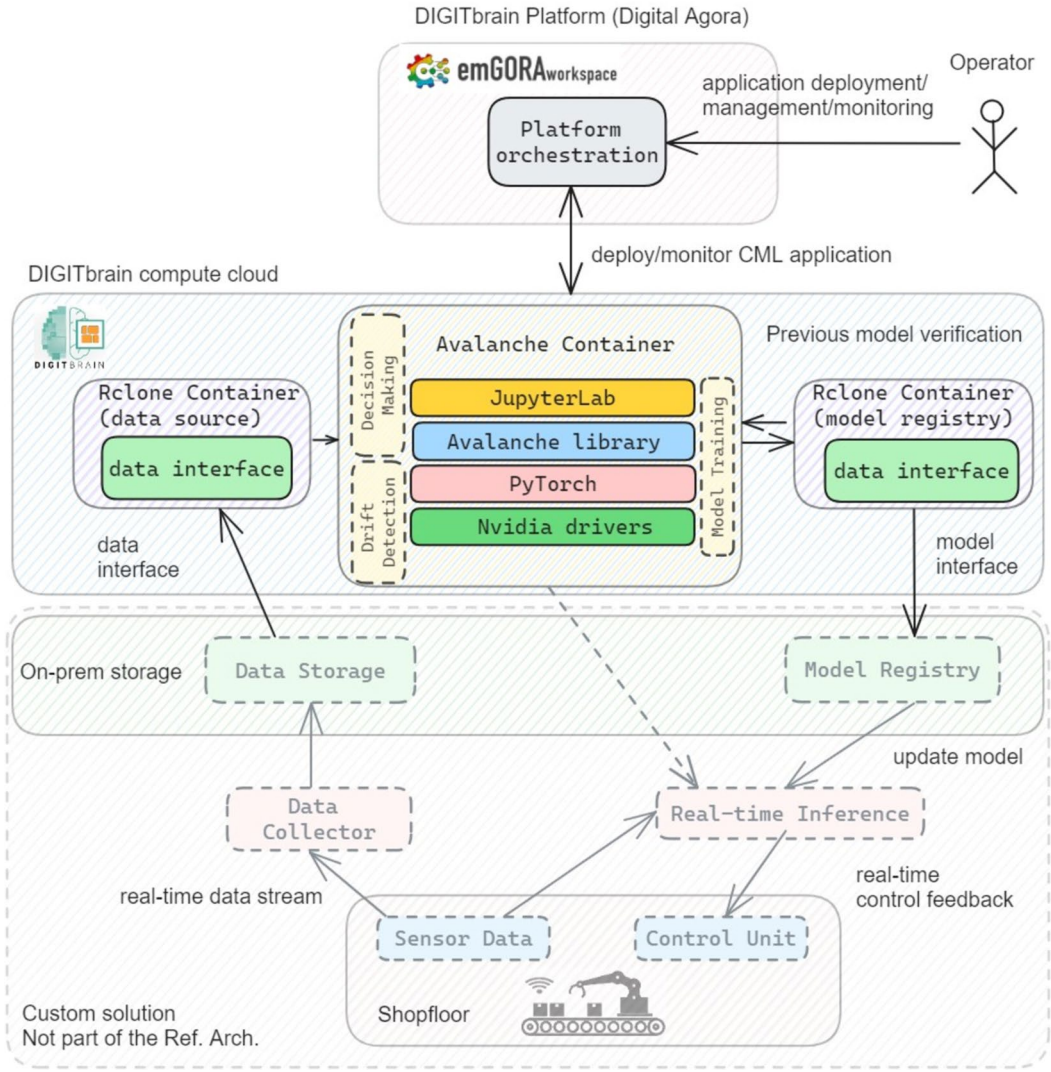

Fig. 2 Concrete CML reference architecture and implementation details

Avalanche container is built on top of the PyTorch stack and extends its functionality to the CL direction. Since, during model development, use of authoring tools (training, evaluating, experimenting, etc.) can be important, we also integrated the widely used JupyterLab, as a graphical development environment option. Moreover, this container is equipped with an NVIDIA driver for GPU execution, which is versatile, allowing testing and utilization of CPU resources whenever GPU resources are not available.

The Avalanche Container allows for realizing the following functionalities: (i) model training and validation; (ii) Decision Making logic that triggers model re-training on extended/re-sampled datasets; (iii) detection of data distribution changes (drift) over time. Since these phases are application-specific, the reference architecture can only provide a framework, tools, and options to implement them but cannot provide hard-wired solutions for all the cases requiring no customization efforts.

Finally, the Reference Architecture when launched by the platform allows users configuring essential parameters such as specifying external data resources (hosts, storage types and data protocols, access credentials, etc., for data and model storages), entering application-specific settings (such as the password to access JupyterLab), as well as keeping track of the

status of current application execution and potential alerts coming from the containers.

Note that the provided concrete reference architecture is an actual implementation (deployable and integrated) and convers and provides frameworks for most of the functional building blocks of the logical reference architecture presented previously. It is not application-specific, so it can be used more generally in various production environments and use cases requiring CML, although it cannot specify details such as how exactly to detect drift, what data/types of data are collected, what particular storage needs to be used and where it is deployed, how to decide when to retrain the model and how, etc., which are application-dependent details. It still offers implementation solutions for several components and frameworks for placeholders subject to specification, customization, tailoring to complete the system.

4 Validation and Experimental Results

The APRICOT experiment (one of the industrial validation experiments in DIGITbrain project) implemented a solution for automatized quality control in the glue application process on parquet manufacturing lines. The end user previously performed quality control using manual methods, which was rather time-consuming, costly and prone to error. To eliminate such bottlenecks and automate the process the experiment utilized high-end, NIR (near-infrared) cameras and deep learning-based computer vision techniques to detect the quality issues in the glue application process on the wooden planks in real time on a dedicated edge device. The images generated from the NIR cameras, as shown in Figure 3, have been utilized to generate an ML model in later stages.

The architecture of the application as shown in Figure 4 consists of multiple microservices:

1. A microservice responsible for image acquisition, image processing and machine learning model inference (*Glue Defect Detection*)
2. A microservice that, depending on the defect detection result, sends a command to a printer to mark the defective plank (PLC Trigger)
3. A microservice for storing collected data in the cloud (*S3 Cloud Storage Microservice*)

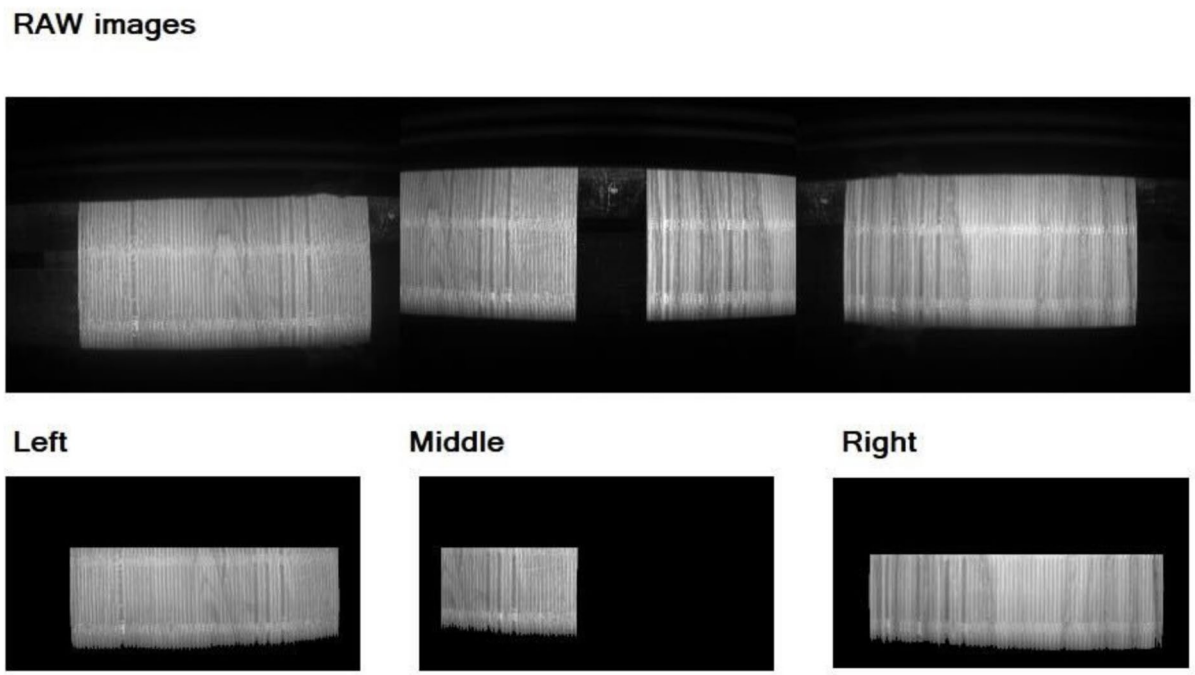

Fig. 3 Image data from the NIR Cameras of the plank surfaces, used as the input data to the ML Model

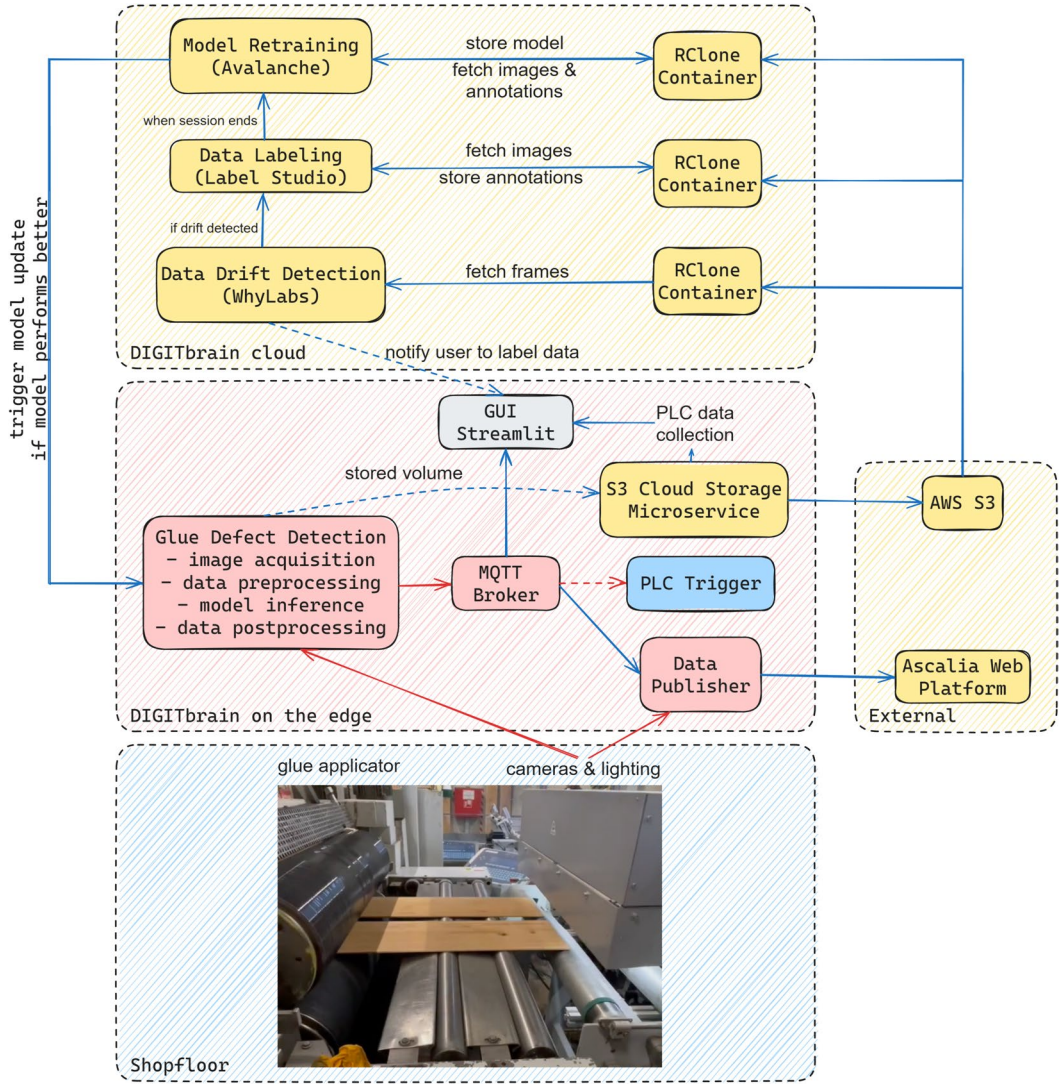

Fig. 4 Architecture of the APRICOT experiment implementation

4. A microservice for publishing metric data to Ascalia's web platform for monitoring production (*Data Publisher*)
5. A graphical user interface showing the data on detected defects and simple statistics
6. An MQTT broker for communication between microservices.

Since the application uses a deep learning model to detect the defects on images, its performance could deteriorate over time due to various factors inherent to industrial environments. Most frequent factors fall into the following categories:

1. **Hardware degradation**: Hardware equipment can degrade over time or stop functioning completely. For instance, lighting intensity could decrease over time, leading to changes in data quality which could affect model performance.
2. **Adaptation challenges**: New product types could be added to production, which could differ from existing data. The ML model in production, which was trained prior to this addition, most likely wouldn't be able to perform well on those new images. To mitigate this additional image data of the new products should be collected and annotated, and the ML model should be retrained.

3. **Environmental changes**: If the environment, in which the hardware setup is installed, undergoes changes (e.g., in lighting conditions, equipment configuration, background), the model could fail to adapt if such conditions weren't represented in the training data.

An increased number of false positive detections could be very disruptive for a manufacturing process, whereas an increased number of false negatives could result in the defect being undetected by the operators, and lead to customer dissatisfaction. Therefore, it's crucial to incorporate data/model monitoring concepts and Continual learning. To implement those concepts into the manufacturing pipeline the current application architecture was extended.

4.1 Challenges

4.1.1 Scarcity of Labeled Data

Typically, in industrial production, defects rarely happen however their significance is huge when they remain undetected and defective products are shipped to the customer. Scarcity of labeled data was a great challenge in this use case. The manufacturing company (targeted user of the application) has not previously used cameras to inspect their products, so they could not provide data. Also, no such open dataset was available at the time of the application development. It required a huge effort to collect, manually (visually) inspect, categorize and verify image labels one-by-one, involving the work of experts, line managers and operators to confirm the actual glue application defects which also depend on parameters such as size, shape and density.

4.1.2 Scenario Complexity

As mentioned previously, changes of the environment can negatively impact the performance of the model, which may then fail to "generalize", detect potential faults under the new conditions. Computer vision applications are sensitive to lighting conditions, and despite that the hardware setup was designed to minimize impact of external lighting conditions, it was impossible to eliminate all interferences. Another challenge was the diversity of plank types to be inspected, which varied in texture and color, furthermore, defects also differed in their shape, size, and frequency, which could result in an imbalanced dataset. The model was expected to perform well on all types of products.

4.1.3 Task-Specific Requirements

The environment can change over time, so as the products and the types of defects being inspected. In addition, after a certain period, the ratio of glue components was updated (to reduce production costs), which resulted in visually different defects. On one product type, there were multiple smaller regions of unapplied glue on planks ("spotty" glue application) and the decision to be reported (defect or not) depended on the distance between the smaller defects (density) and their size.

Considering the above challenges, continual learning was a critical aspect at developing such a computer vision application.

4.2 Application Overview

A microservice for data drift detection was developed to detect changes in the acquired images. The microservices utilizes WhyLabs [66], an open-source data logging platform, to detect drift in the input data. Using a reference dataset, the platform performs different statistical or machine-learning-based methods to detect changes in the data distributions. When a drift is detected, the microservice notifies the operator about the change on a graphical user interface, which recommends reviewing and annotating the new data. Also, it can trigger immediate interactions, interventions in case of sudden drops, unexpected behavior of the production machine or unusual change of the environmental conditions.

The operator is then directed to use the data annotation microservice, another graphical user interface wrapping LabelStudio [67], an open-source platform for data labeling. Using this microservice, the user can annotate previously collected image data from the cameras. The annotations are automatically stored in an S3 bucket (AWS cloud storage solution).

Once the data labeling session has been completed, the microservice responsible for retraining

models is notified, and training jobs are started. The resulting model is evaluated and stored in S3. If the new model performs better, the glue defect detection microservice fetches the new model automatically. The following paragraphs describe in detail the drift detection and retraining microservices that have been implemented.

4.3 Data Drift Detection

To assess data drift the WhyLabs platform initiates a profiling process on the data before the drift detection algorithm is applied. These profiles encapsulate essential statistical attributes of the data, including distribution metrics, frequent items, and the identification of missing values. Such statistical properties are then employed in modified versions of drift detection techniques. It is crucial to acknowledge that a profile is merely an estimation of the original data. Its use for drift detection results in approximations rather than precise identification of the actual drift [68].

Various methodologies exist in drift detection to quantify the similarity between two probability distributions or data profiles. The selection of a specific method hinges on the unique requirements of the use case. The WhyLabs Platform accommodates four algorithms for drift detection and monitoring: Hellinger distance, Kullback-Leibler (KL) divergence, Jensen-Shannon (JS) divergence, and Population Stability Index (PSI). Each algorithm has distinct advantages and disadvantages, necessitating some trial-and-error to fine-tune the monitoring process to attain an optimal signal-to-noise ratio.

The default choice for calculating drift is the Hellinger distance due to its applicability to discrete (categorical) and non-discrete (numerical) features and its interpretability. This metric, derived from comparing the square root of probability distributions, yields values between 0 and 1. A value of 0 signifies no divergence, while a value of 1 denotes entirely distinct distributions. However, its robustness may limit its sensitivity to subtle changes in distribution, making alternatives like KL or JS divergence more suitable in certain cases [69].

Monitoring unstructured data, such as images, and telemetry data, is captured in a structured format compatible with conventional statistical approaches. For instance, monitoring changes in image brightness can be achieved by calculating the mean pixel value, while hue and saturation offer insights into the image color palette. Monitoring image height involves capturing the shape of the tensor representing raw image data along with the number of channels in the tensor indicating the image's color space (RGB, CMYK, grayscale) [70].

In the reference example, parameters such as image height, width, color space, brightness, and entropy (representing image complexity) are continuously monitored. Once the divergence between reference and current profiles surpasses a predefined threshold, indicating a drift, the user is promptly notified through the user interface to review and potentially annotate new data for retraining purposes.

4.4 Model Retraining Microservice

The process of retraining a complex model from scratch whenever new data is available can be extremely costly in terms of storage and compute resources, even when conducted on servers [71]. Therefore, the reference architecture's model retraining service utilizes the Python library mentioned earlier, Avalanche for rehearsal strategy implementation, a simple continual learning strategy consisting of learning new experiences as well as re-learning/repeating those that were already seen by the deployed model to minimize the effects of catastrophic forgetting and reduce the training complexity. Previously learned experiences are available and can be fetched from a rehearsal buffer, a dynamic dataset (with a fixed maximum size) that can be continuously updated with new data as needed. This approach employs a custom data loader that fetches previous experiences from the rehearsal buffer and injects them into each training batch composed primarily of new experiences.

Avalanche proved to be a good fit for this application because it is an easy-to-use, open-source Python library that extends PyTorch with continual learning capabilities. It covers the whole process, from data loading to model training and evaluation. Since it is designed for fast prototyping, adjusting the training loop for the PyTorch-based machine learning model was not difficult. The updated versions of the model are stored in AWS S3 cloud storage, utilizing its versioning feature. All versions are kept alongside the

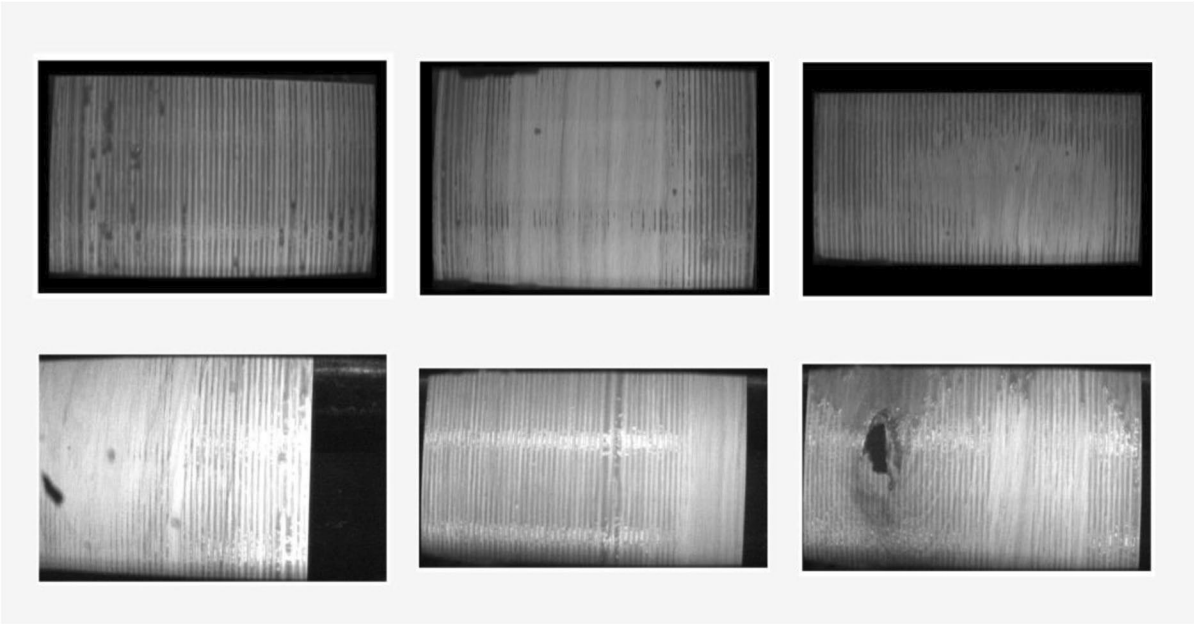

Fig. 5 Input data evolution: images taken before (top row) and after (bottom row) environmental changes. A noticeable increase in brightness and the presence of reflections can be observed

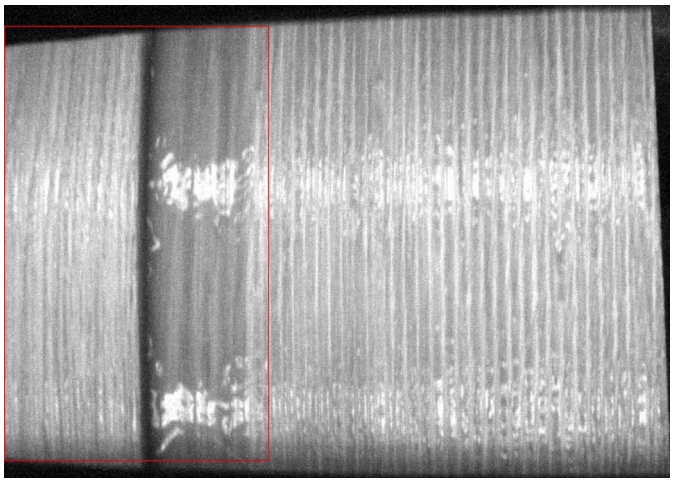

Fig. 6 False positive detection made by the initial model on the latest input images due to data drift

model's metadata, in case the updated model's performance has not improved, necessitating a rollback.

4.5 Experimental Results

During the initial model development phase, an object detection model (YOLO architecture) was trained to identify defects in the glue application process using a dataset composed of 300 initial images.

However, after the model was deployed into production, the environment changed slightly due to decaying lighting conditions. Even though the cameras and lighting were placed in the industrial casing to minimize external influences, the housing might have been readjusted, causing changes in how the lighting reflects from the plank surfaces. It can be observed from the image comparison in Figure 5 that the latest images had higher brightness and

Table 1 Comparison of metrics between the initial and updated model on validation datasets before and after environmental changes

	Precision	Recall	F1 Score	mAP50	mAP50-95
Initial dataset	0.985	0.792	0.878	0.873	0.581
Updated dataset	0.957	0.936	0.946	0.957	0.871

visible reflections. Consequently, the initial dataset was updated to account for these environmental changes.

Occurrences of false positive detection, as depicted in Figure 6, have escalated during production. To address this the model was updated using the replay rehearsal strategy within Avalanche, as described in the previous subsection, with default hyperparameters. In the updated dataset, there were 254 new images in total (204 used for training and 50 for testing). Training the model required about 20 minutes on a ml.p2.xlarge GPU instance on AWS.

The comparison of model performances shown in Table 1 illustrates the impact of updating the model with a continual learning strategy on various performance metrics. The precision dropped slightly (from 0.985 to 0.957); however, it could also be seen that recall improved significantly (from 0.792 to 0.936). To clients in this industry that means that much less defects were missed by the quality control system, while there was a slight increase in false positive detections. As it was previously mentioned, the error of missing a true positive defect is much greater to the manufacturing company than having a false positive detection. In this use case especially, since after the glue application phase the defects cannot be detected in any way due to the nature of the process, and the consequences become visible only after the parquet is shipped to the end user, which is costly for the company and damaging to the reputation.

Having false positive detections could impact the production process in a way that operators spend too much time on false detections or increase costs due to larger amounts of planks being withdrawn from production. However, the false positives in this case occur rarely enough, and when they occur, the results can be used to further improve the model. Operators at the manufacturing line have a user interface that notifies them about the defects with visualization. If the reported defect is false positive, the operators can report it as a false positive detection using a button. That sample is stored in the dataset for later retraining and validation purposes. The operators or other personnel with expertise on quality control have the ability to label the images using the application, which then triggers the model retraining pipeline and deploys the new model in case of better performance.

Therefore, even though precision dropped slightly, the updated model performs better overall and can be deployed into production. Slightly increased false positive detection rate is not disruptive to the production, and can only benefit from the continual learning process, since images will be annotated and used in future training. Due to the nature of the framework, the majority of the pipeline is automatized so the end user is only required to interact with the user interface and annotate images in case of drift detection or false positive detections.

5 Discussion and Conclusions

Continual Machine Learning ensures dynamic adaptation of ML models to these new conditions by monitoring and re-fitting models according to the shift or more radical changes, from time to time. To implement such a continuously evolving system is not a straightforward task due to its complexity.

The paper overviewed state-of-the-art techniques on how to recognize, identify change points, characterize and quantify concept drift (based on error rate, data distribution or on a combination of both) as well approaches to how to adapt and react upon these changes (by retraining models or constructing ensemble models). We provided a high-level, logical architecture that includes all the necessary functional building blocks, their relationships and the main workflows (“hot” and “cold” data paths), keeping implementation decisions open (concrete software choices, location of deployments, integration questions), which can serve as a reference or simply as a guide for the more detailed system design. The paper proposed a particular implementation for this logical reference architecture (centered around Avalanche, based on PyTorch, and with data interfaces through Rclone), which was then further customized and successfully validated in an industrial application at the

industrial plant of APRICOT showing that the presented approach has also practical usability.

This research proposed one realization for the high-level Continual machine learning reference architecture (based on Avalanche and PyTorch), however, other implementations may also prove to be widely usable in different domains in the industry, potentially using other CML toolkits and other machine learning frameworks (Keras-TensorFlow, Scikit Learn, etc.). The experimental validation further narrowed the focus to specific drift detection and decision-making strategies (optimized accordingly to the objectives of the current manufacturing process), furthermore, the implementation was not fully automated; human-in-the-loop was required to assist in labelling new data to revise or override decisions made by the system. The investigation of these options and further improvements serve as a basis for our future work.

Acknowledgements Ascalia would like to acknowledge their colleague Valentino Petric, who was instrumental in the development of the initial machine learning model and its deployment, which was anything but an easy task. Furthermore, they extend their gratitude to Boris Poklepovic and his team at Bauwerk Group Hrvatska for providing the opportunity to address the problem in industry at´ their factory and for tirelessly responding to inquiries with their expertise and readiness to assist.

Author Contributions Conceptualization: J.A., Á.H. and M.E.; Methodology: J.A., S.B., E.M. and A.C.M.; Software: M.E., D.J., V.Ž. and D.S.; Writing—original draft: J.A., D.J., Á.H., V.Ž., M.E., D.S. and A.C.M.; Writing—reviewing and editing: J.A., D.J., Á.H., V.Ž., T.L. and M.E.; Supervision: S.B., T.L. and A.C.M.; Validation: D.J., V.Ž. and D.S.

Funding This work was funded by European Union's Horizon 2020 project titled:"Digital twins bringing agility and innovation to manufacturing SMEs, by empowering a network of DIHs with an integrated digital platform that enables Manufacturing as a Service (MaaS)" (DIGITbrain), under grant agreement no. 952071. This work also received financing from the Hungarian project no. TKP2021-NVA-01, implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme. This work was also sponsored/funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) in the context of the project "Digitaler Zwilling und Künstliche Intelligenz in der vernetzten Fabrik für die integrierte Nutzfahrzeugproduktion, Logistik und Qualitätssicherung" (TWIN-4TRUCKS, 13IK010F). The authors are grateful for this support.

Data Availability The datasets generated and analyzed during the use case are not publicly available due to company privacy reasons (Ascalia-Bauwerk proprietary data). The main contributions of the paper (conceptual and concrete reference architectures, techniques in the field of continual learning) however do not depend on neither a direct consequence of the particular data used but can more generally be applied.

Declarations

Ethical Approval Not applicable.

Competing Interests The authors declare no competing interests.

References

1. Matrenin, P., Antonenkov, D., Arestova, A.: Energy efficiency improvement of industrial enterprise based on machine learning electricity tariff forecasting. In: Proceedings of the 2021 XV international scientific-technical conference on actual problems of electronic instrument engineering (APEIE), pp. 185–189. IEEE (2021). https://doi.org/10.1109/APEIE52976.2021.9647491
2. Szott, S., et al.: Wi-fi meets ml: A survey on improving ieee 802.11 performance with machine learning. IEEE Commun. Surv. Tutorials **24**, 1843–1893 (2022)
3. Lin, X., Bogdan, P., Chang, N., Pedram, M.: Machine learning-based energy management in a hybrid electric vehicle to minimize total operating cost. In: 2015 IEEE/ACM international conference on computer-aided design (ICCAD), pp. 627–634, Austin, TX (2015). https://doi.org/10.1109/ICCAD.2015.7372628
4. Osypanka, P., Nawrocki, P.: Resource usage cost optimization in cloud computing using machine learning. IEEE Trans. Cloud Comput. **10**, 2079–2089 (2020)
5. Perera, A.D., Jayamaha, N.P., Grigg, N.P., Tunnicliffe, M., Singh, A.: The application of machine learning to consolidate critical success factors of lean six sigma. IEEE Access **9**, 112411–112424 (2021)
6. Chatterjee, S., Misbahuddin, M., Vamsi, P., Ahmed, M.H.: Power quality improvement and fault diagnosis of PV

system by machine learning techniques. In: 2023 international conference on signal processing, computation, electronics, power and telecommunication (IConSCEPT), pp. 1–6, Karaikal (2023). https://doi.org/10.1109/IConSCEPT57958.2023.10170117
7. Paolanti, M., Romeo, L., Felicetti, A., Mancini, A., Frontoni, E., Loncarski, J.: Machine learning approach for predictive maintenance in industry 4.0. In: 2018 14th IEEE/ASME international conference on mechatronic and embedded systems and applications (MESA), pp. 1–6, Oulu (2018). https://doi.org/10.1109/MESA.2018.8449150
8. Susto, G.A., Schirru, A., Pampuri, S., McLoone, S., Beghi, A.: Machine learning for predictive maintenance: A multiple classifier approach. IEEE Trans. Industr. Inf. **11**, 812–820 (2014)
9. Ceesay, R., Boonchoo, T., Rattanatamrong, P.: Machine learning approaches for quality control in pulp packaging manufacturers. In: 2023 20th international joint conference on computer science and software engineering (JCSSE), pp. 385–390, Phitsanulok (2023). https://doi.org/10.1109/JCSSE58229.2023.10202113
10. Peres, R.S., Barata, J., Leitao, P., Garcia, G.: Multistage quality control using machine learning in the automotive industry. IEEE Access **7**, 79908–79916 (2019)
11. Sun, S., Cao, Z., Zhu, H., Zhao, J.: A survey of optimization methods from a machine learning perspective. IEEE Trans. Cybern. **50**, 3668–3681 (2019)
12. Suzuki, Y., Iwashita, S., Sato, T., Yonemichi, H., Moki, H., Moriya, T.: Machine learning approaches for process optimization. In: 2018 international symposium on semiconductor manufacturing (ISSM), pp. 1–4, Tokyo (2018). https://doi.org/10.1109/ISSM.2018.8651142
13. Dhiman, D., Bisht, A., Kumari, A., Anandaram, D.H., Saxena, S., Joshi, K.: Online fraud detection using machine learning. In: 2023 international conference on artificial intelligence and smart communication (AISC), pp. 161–164, Greater Noida (2023). https://doi.org/10.1109/AISC56616.2023.10085493
14 Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., Wermter, S.: Continual lifelong learning with neural networks: A review. Neural Netw: Off. J. Int. Neural Netw. Soc. **113**, 54–71 (2019)
15. French, R.: Catastrophic forgetting in connectionist networks. Trends Cogn. Sci. **3**, 128–135 (1999)
16. Grossberg, S.: Studies of mind and brain: neural principles of learning, perception, development, cognition, and motor control, vol. 70 of Boston studies in the philosophy of science. Springer, Dordrecht (1982). https://doi.org/10.1007/978-94-009-7758-7
17. Abraham, W.C., Robins, A.: Memory retention–the synaptic stability versus plasticity dilemma. Trends Neurosci. **28**, 73–78 (2005)
18. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. In Bower, G. H. (ed.) *The psychology of learning and motivation*, vol. 24 of Psychology of Learning and Motivation, 109–165 (Academic Press, London). (1989)
19. Thrun, S., Mitchell, T.M.: Lifelong robot learning. In: Steels, L. (ed.) The biology and technology of intelligent autonomous agents. NATO ASI Series, vol 144. Springer, Berlin, Heidelberg (1995). https://doi.org/10.1007/978-3-642-79629-6_7
20. Hassabis, D., Kumaran, D., Summerfield, C., Botvinick, M.: Neuroscience-inspired artificial intelligence. Neuron **95**, 245–258 (2017)
21. DIGITbrain: Horizon 2020 project (2020). https://digitbrain.eu/. Accessed 3 Mar 2023.
22. Deslauriers, J., Kiss, T., Kovacs, J.: Dynamic composition and automated deployment of digital twins for manufacturing. In: Proceedings of the 13th international workshop on science gateways, CEUR workshop proceedings (2021)
23 Zambrano, V., et al.: Industrial digitalization in the industry 4.0 era: Classification, reuse and authoring of digital models on digital twin platforms. Array **14**, 100176 (2022)
24. Schweichhart, K.: Reference architectural model industrie 4.0 (RAMI 4.0). (2016). https://ec.europa.eu/futurium/en/system/files/ged/a2-schweichhart-reference_architectural_model_industrie_4.0_rami_4.0.pdf. Accessed 14 Nov 2024.
25. Lin, S.-W., Miller, B., Durand, J., Joshi, J., Didier, P., Chigani, P., Torenbeek, R., Duggal, D., Martin, R., Bleakley, G.: Industrial internet reference architecture. In: Industrial Internet Consortium (IIC). Tech. Rep (2015)
26. Amazon Web Services: AWS Architecture Center—Architecture Best Practices for Analytics & Big Data. https://aws.amazon.com/architecture/analytics-big-data/. Accessed 5 November 2023.
27. Microsoft Azure IoT — Internet of Things Platform. https://azure.microsoft.com/en-us/solutions/iot. Accessed 5 November 2023.
28. Marosi, A.C., et al.: Interoperable data analytics reference architectures empowering digital-twin-aided manufacturing. Futur. Internet **14**, 114 (2022)
29. Marosi, A.C., et al.: Toward reference architectures: A cloud-agnostic data analytics platform empowering autonomous systems. IEEE Access **10**, 60658–60673 (2022)
30. Pierantoni, G., Kiss, T., Bolotov, A., Kagialis, D., DesLauriers, J., Ullah, A., Chen, H., Fee, D.C.Y., Dang, H.V., Kovacs, J., Belehaki, A., Herekakis, T., Tsagouri, I., Gesing, S.: Toward a reference architecture based science gateway framework with embedded e-learning support. Concurr. Comput. Pract. Exp. **35**(18), (2023). https://doi.org/10.1002/cpe.6872
31. Farkas, Z., Lovas, R.: Reference architecture for IOT platforms towards cloud continuum based on Apache Kafka and orchestration methods. In: Proceedings of the 7th international conference on internet of things, big data and security (IoTBDS), pp. 205–214 (2022). https://doi.org/10.5220/0011071300003194
32. Pallathadka, H., et al.: A review of using artificial intelligence and machine learning in food and agriculture industry. In 2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 2215–2218, https://doi.org/10.1109/ICACITE53722.2022.9823427 (2022)
33. Qin, J., et al.: A machine learning methodology for diagnosing chronic kidney disease. IEEE Access **8**, 20991–21002 (2020). https://doi.org/10.1109/ACCESS.2019.2963053

34. Xin, Y., et al.: Machine learning and deep learning methods for cybersecurity. IEEE Access **6**, 35365–35381 (2018). https://doi.org/10.1109/ACCESS.2018.2836950
35. Khayyam, H., et al.: A novel hybrid machine learning algorithm for limited and big data modeling with application in industry 4.0. IEEE Access **8**, 111381–111393 (2020). https://doi.org/10.1109/ACCESS.2020.2999898
36. Žliobaite, I.: Learning under concept drift: an overview. arXiv preprint arXiv:1010.4784. (2010). https://doi.org/10.48550/arXiv.1010.4784
37. Lu, J., et al.: Learning under concept drift: A review. IEEE Trans. Knowl. Data Eng. **31**, 2346–2363 (2019). https://doi.org/10.1109/TKDE.2018.2876857
38. Wang, H., Abraham, Z.: Concept drift detection for streaming data. In: 2015 international joint conference on neural networks (IJCNN), pp. 1–9, Killarney (2015). https://doi.org/10.1109/IJCNN.2015.7280398
39. Žliobaitė, I., Pechenizkiy, M., Gama, J.: An overview of concept drift applications. In: Japkowicz, N., Stefanowski, J. (eds.) Big data analysis: new algorithms for a new society. Studies in big data, vol. 16. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-26989-4_4
40. Basseville, M., Nikiforov, I.V.: Detection of abrupt changes - theory and application. Prentice Hall, Inc (1993)
41. Gama, J., Castillo, G.: Learning with local drift detection. In: Li, X., Zaïane, O.R., Li, Z. (eds.) Advanced data mining and applications. ADMA 2006. Lecture notes in computer science(), vol. 4093. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11811305_4
42. Gama, J., Castillo, G.: Learning with local drift detection. In International conference on advanced data mining and applications, 42–55 (Springer). (2006)
43. Baena-Garcıa, M., et al.: Early drift detection method. In Fourth international workshop on knowledge discovery from data streams, vol. 6, 77–86 (Citeseer). (2006)
44. Xu, S., Wang, J.: Dynamic extreme learning machine for data stream classification. Neurocomputing **238**, 433–449 (2017). https://doi.org/10.1016/j.neucom.2016.12.078
45. Alippi, C., Roveri, M.: Just-in-time adaptive classifiers—part i: Detecting nonstationary changes. Trans. Neur. Netw. **19**, 1145–1153 (2008). https://doi.org/10.1109/TNN.2008.2000082
46. Alippi, C., Roveri, M.: Just-in-time adaptive classifiers—part ii: Designing the classifier. Trans. Neur. Netw. **19**, 2053–2064 (2008). https://doi.org/10.1109/TNN.2008.2003998
47. Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowing. In Proceedings of the 2007 SIAM international conference on data mining, 443–448 (SIAM). (2007)
48. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of the 7th SIAM international conference on data mining, vol. 7 (2007). https://doi.org/10.1137/1.9781611972771.42
49. Jourdan, N., Bayer, T., Biegel, T., Metternich, J.: Handling concept drift in deep learning applications for process monitoring. Procedia CIRP. 56th CIRP International Conference on Manufacturing Systems 2023. **120**, 33–38 (2023). https://doi.org/10.1016/j.procir.2023.08.007
50. Kvaktun, D., Liu, D. & Schiffers, R. Detection of concept drift for quality prediction and process control in injection molding. AIP Conf. Proc. **2884**(1), (2023). https://doi.org/10.1063/5.0168491
51. Lange, M.D., et al.: Continual learning: A comparative study on how to defy forgetting in classification tasks. https://doi.org/10.48550/arXiv.1909.08383
52. Mirzadeh, S. I., Farajtabar, M., Pascanu, R., Ghasemzadeh, H.: Understanding the role of training regimes in continual learning. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems, vol. 33, 7308–7320 (Curran Associates, Inc.,). (2020)
53. Li, Z., Hoiem, D.: Learning without forgetting. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) Computer vision - 14th European conference, ECCV 2016, proceedings (pp. 614-629). (Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics); Vol. 9908 LNCS). Springer (2016). https://doi.org/10.1007/978-3-319-46493-0_37
54. Jung, H., Ju, J., Jung, M., Kim, J.: Less-forgetting learning in deep neural networks. https://doi.org/10.48550/arXiv.1607.00122
55. Maltoni, D., Lomonaco, V.: Continuous learning in single-incremental-task scenarios. https://doi.org/10.48550/arXiv.1806.08568
56. Rusu, A.A., et al.: Progressive neural networks. https://doi.org/10.48550/arXiv.1606.04671
57. Draelos, T.J. et al.: Neurogenesis deep learning: Extending deep networks to accommodate new classes. In 2017 International Joint Conference on Neural Networks (IJCNN), 526–533 (2017). https://doi.org/10.1109/IJCNN.2017.7965898
58. Gepperth, A., Karaoguz, C.: A bio-inspired incremental learning architecture for applied perceptual problems. Cogn. Comput. **8**, 924–934 (2016)
59. Lopez-Paz, David and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. Neural information processing systems (2017).
60. Rajasegaran, J., Khan, S., Hayat, M., Khan, F.S., Shah, M.: iTAML: An incremental task-agnostic meta-learning approach. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 13585–13594 (2020). https://doi.org/10.1109/CVPR42600.2020.01360
61. Tang, S., Chen, D., Zhu, J., Yu, S., Ouyang, W.: Layer-wise optimization by gradient decomposition for continual learning. In: 2021 IEEE/CVF conference on computer vision and pattern recognition (CVPR), pp. 9629–9638, Nashville, TN (2021). https://doi.org/10.1109/CVPR46437.2021.00951
62. Saha, G., Garg, I., Roy, K.: Gradient projection memory for continual learning. https://doi.org/10.48550/arXiv.2103.09762
63. Wang, L., Zhang, X., Su, H., Zhu, J.: A comprehensive survey of continual learning: theory, method and application. (2023). https://doi.org/10.48550/arXiv.2302.00487
64. Zaharia, M., et al.: Accelerating the machine learning lifecycle with mlflow. IEEE Data Eng. Bull. **41**, 39–45 (2018)
65. Lomonaco, V., et al.: Avalanche: an end-to-end library for continual learning. In: Proceedings of the 2021 IEEE/CVF conference on computer vision and pattern recognition

workshops (CVPRW), pp. 3595–3605 (2021). https://doi.org/10.1109/CVPRW53098.2021.00399
66. WhyLabs: "whylogs", GitHub repository, version 1.3.10. https://github.com/whylabs/whylogs. Accessed 21 Nov 2023
67. HumanSignal: "Label studio", an open source data labeling platform, version 1.9.2. https://labelstud.io. Accessed 6 Dec 2023
68. WhyLabs: Whylabs documentation: Profile overview. https://docs.whylabs.ai/docs/overview-profiles/. Accessed 8 Dec 2023
69. WhyLabs: Whylabs documentation: Supported drift algorithms. https://docs.whylabs.ai/docs/drift-algorithms/. Accessed 5 Dec 2023
70. WhyLabs: Whylabs documentation: Image data. https://docs.whylabs.ai/docs/image-data/. Accessed 8 Dec 2023
71. Pellegrini, L., Graffieti, G., Lomonaco, V., Maltoni, D.: Latent replay for real-time continual learning. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 10203–10209 (2020). https://doi.org/10.1109/IROS45743.2020.9341460
72. Kreuzberger, D., Kühl, N., Hirschl, S.: Machine learning operations (mlops): Overview, definition, and architecture. IEEE Access **11**, 31866–31879 (2023)
73. Lacson, R., Eskian, M., Licaros, A., Kapoor, N., Khorasani, R.: Machine learning model drift: predicting diagnostic imaging follow-up as a case example. J. Am. Coll. Radiol. **19**(10), 1162–1169 (2022). https://doi.org/10.1016/j.jacr.2022.05.030