Scanflow-K8s: Agent-based Framework for Autonomic Management and Supervision of ML Workflows in Kubernetes Clusters

Peini Liu*†, Gusseppe Bravo-Rocca*†, Jordi Guitart*†,
Ajay Dholakia‡, David Ellison‡, Miroslav Hodak ‡
*Barcelona Supercomputing Center, Barcelona, Spain
†Universitat Politècnica de Catalunya, Barcelona, Spain
‡Lenovo Infrastructure Solutions Group, Lenovo, Morrisville, NC, USA
E-mail: {peini.liu, gusseppe.bravo, jordi.guitart}@bsc.es,{adholakia,dellison,mhodak}@lenovo.com

***Abstract*—Machine Learning (ML) projects are currently heavily based on workflows composed of some reproducible steps and executed as containerized pipelines to build or deploy ML models efficiently because of the flexibility, portability, and fast delivery they provide to the ML life-cycle. However, deployed models need to be watched and constantly managed, supervised, and debugged to guarantee their availability, validity, and robustness in unexpected situations. Therefore, containerized ML workflows would benefit from leveraging flexible and diverse autonomic capabilities. This work presents an architecture for autonomic ML workflows with abilities for multi-layered control, based on an agent-based approach that enables autonomic management and supervision of ML workflows at the application layer and the infrastructure layer (by collaborating with the orchestrator). We redesign the Scanflow ML framework to support such multi-agent approach by using triggers, primitives, and strategies. We also implement a practical platform, so-called Scanflow-K8s, that enables autonomic ML workflows on Kubernetes clusters based on the Scanflow agents. MNIST image classification and MLPerf ImageNet classification benchmarks are used as case studies to show the capabilities of Scanflow-K8s under different scenarios. The experimental results demonstrate the feasibility and effectiveness of our proposed agent approach and the Scanflow-K8s platform for the autonomic management of ML workflows in Kubernetes clusters at multiple layers.**

***Index Terms*—Scanflow, Machine Learning Workflow, Autonomic, Self-Management, Agent, Kubernetes, MLOps**

I. Introduction

Machine Learning (ML) has become common with good results in different tasks such as image classification, machine translation, recommendation systems, and speech recognition. While working on a ML project, workflows comprising some reproducible steps run as a pipeline are widely used to build or deploy a model efficiently because of the flexibility, portability, and fast delivery they provide to the ML life-cycle [1].

ML workflows still face several challenges while being used by different teams. The Data Science team requires to automate some repetitive tasks within ML workflows to train and improve the model [2], [3]. Therefore, some AutoML modules and frameworks have been developed for algorithm selection [4], model selection [5], and feature selection [6] to tune hyperparameters and have good learning performance with less human assistance. However, ML life-cycle is more than just training a model [7]. Once the model has been trained, the Data Engineer team works on deploying the ML workflows into production. More importantly, they are required to operate the workflows to maintain the robustness of the model at runtime, that is, to deal with security vulnerabilities, concept drift, lack of explainability and interpretability, and hidden technical debt [8], [9], because the model may degrade its accuracy due to constantly evolving data profiles. Also, the model online inference serving services have strict latency requirements and efficiency issues that should be considered [10]–[12]. Therefore, the ML workflow is no longer running in a known context and with static requirements, and consequently, enabling the autonomy to manage and supervise ML workflows to meet dynamic changes has become an open issue [13].

Autonomic computing brings inspiring approaches to adapt ML systems at runtime, helping to manage and supervise the ML workflows operation in dynamic contexts [14]–[17]. For example, by enabling adaptive learning algorithms for streaming data to supervise ML models at the application layer or by reconfiguring and restructuring the workflows at the infrastructure layer [18]. Consequently, our work enables an agent-based approach to leverage autonomic computing for ML workflows system to meet dynamic changes. The agents focus on the robustness and requirements of the model at the application layer while managing the quality of services and the structure of workflows at the infrastructure layer.

In our previous work, we presented Scanflow, an executor multi-graph framework for end-to-end ML workflow management and debugging in an offline mode, in the form of a proof-of-concept prototype running in a single node, which featured an anomaly detector of out-of-distribution samples in the inference phase [19]. In this paper, we contribute Scanflow-K8s, a functional agent-based MLOps framework that enables autonomic management and online supervision of the end-to-end life-cycle of ML workflows on Kubernetes. Scanflow-K8s redesigns Scanflow from scratch to upgrade the executor nodes to a multi-agent system based on triggers, primitives, and strategies, and to be fully integrated with the Kubernetes platform, enabling autonomic multi-layer management and supervision of ML workflows in clusters.

The remainder of this paper is organized as follows: Section 2 discusses the related work. Section 3 introduces the multiple management layers in autonomic ML workflows. Section 4 describes the agent architecture, social ability, triggers, and operation primitives. Section 5 presents some case studies and experiments on Scanflow-K8s platform. Finally, Section 6 concludes the paper and discusses the future work.

II. Related Work

Lately, many data researchers and companies have been interested in automating the ML tasks within a training workflow (e.g., AutoML) in order to construct ML models efficiently [2]–[6]. However, these powerful AutoML modules and frameworks (e.g., Kubeflow Pipelines[1]) are turned off after training a model, thus cannot help the model after being deployed to meet dynamic changes.

To make an autonomic system for ML, Kedziora et al. [13] defined an autonomous system (i.e., AutonoML) as one showing fundamental characteristics of persistence and adaptation. Persistence means that an AutonoML system should be capable of operations in the long term, and adaptation refers to the theories and practices of facing dynamic contexts. Zliobaite et al. [18] identified challenges in designing and building adaptive learning (prediction) systems to achieve scalability, usability, and trust, taking into account various application needs. These works provide a conceptual level view or framework without practical implementation or evaluation.

Seldon[2] provides a set of tools for deploying ML models at scale and presents their practical oversight and governance for ML deployments. But these tools (so-called Alibi) mainly focus on metrics monitoring, outliers and drift detection, and model explanation [20], rather than autonomically taking actions to maintain the model performance. KubeDL[3] supports running different deep learning workloads on Kubernetes. It considers training, model version, model serving, and also an auto-configuration framework Morphling [21] to tune the best configuration before the serving service is deployed. However, the training steps are considered as jobs and the model serving is considered as a simple service rather than a ML workflow, losing the flexibility of using workflows, and also autonomy is not considered. KServe[4], formerly KFServing and used by Kubeflow, enables serverless model inference on Kubernetes. It encapsulates the complexity of autoscaling, networking, health checking, and server configuration to bring serving features like GPU autoscaling, scale to zero, and canary rollouts to the ML deployments. However, it is based on the serverless model supported by Knative, which can only support streaming online inference. Moreover, it integrates Alibi addons to detect anomalies, but not deal with them autonomically.

[1]https://www.kubeflow.org/
[2]https://www.seldon.io/
[3]https://kubedl.io/
[4]https://kserve.github.io/website

Some adaptive learning algorithms are designed for streaming (unpredicted new data arrives) [8], [9], [22]. Gama et al. [8] presented a survey on concept drift adaptation, which introduces the online adaptive learning processes and algorithms. Imbrea [22] proposed a framework for implementing AutoML on data streams architectures in production and indicated that, in the presence of concept drift, detection or adaptation techniques have to be applied to maintain the predictive accuracy over time. These adaptive learning systems or methods can deal with partially dynamic contexts, but as we discussed in the introduction, autonomy should be applied at multiple levels to handle both robustness and efficiency problems.

Autonomic computing theories and practices have been used in multiple areas. Formerly, they were applied in the service-oriented computing paradigm [17]. Lately, as systems were adopting the microservice architecture, Liu et al. [14] studied the autonomy in those microservice-based systems. Also, some works showed the usage of agents for autonomic computing [15]–[17]. These related works did not directly show how to bring autonomy for ML workflows, but they inspired our work for adopting an agent-based autonomic approach.

III. Architecture for Autonomic ML Workflows

In this section, we firstly describe diverse uncertainties that occur in ML workflows. Then, we present an architecture for autonomic ML workflows featuring a multi-layered autonomic framework. Finally, we present a practically implemented platform that enables autonomic ML workflows on Kubernetes clusters based on agents.

A. Uncertainties in ML Workflows

The need to embed ML systems into long-lived dynamic contexts is likely to increase in the coming years [13], thus inherent uncertainties in ML workflows may increase when they are deployed in production (e.g., Cloud). Table I shows a taxonomy of potential uncertainties in ML workflows.

TABLE I
Uncertainties in ML workflows

Categories		Examples
Requirements	Functional Requirements	End-users expect robust ML models when facing data drift; Data scientist adds new steps to the workflow
	Non-functional Requirements	End-users define some QoS requirement for model serving; Data engineer provides resource restrictions and affinity settings for workflow executors
Contexts	Workflow Contexts	Workflow executor fails to run (e.g., software bug, out-of-memory error); Model serving service is not available
	System Contexts	Other workflows compete to use the same resources
	External Contexts	Hardware/Operating System crashes or is not available

Uncertainties in ML workflows come from two main sources, namely the requirements and the context. The former come from the data scientists and end-users and include the functional and non-functional requirements of the ML

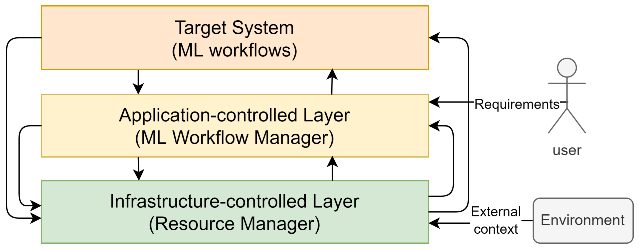

Fig. 1. Conceptual architecture of multi-layer controlled ML workflows.

workflows. In particular, (1) the functional requirements can include changes in the topology of the ML workflows (e.g., adding new steps), as well as the need to maintain the quality and robustness of the ML models (e.g., dealing with security vulnerabilities, outliers, concept drift, and model explainability) in unexpected situations; (2) the non-functional requirements can include the need to fulfill the QoS (Quality of Service) guarantees related to the model serving service's performance (e.g., latency and failure rate) in the occurrence of churn, as well as the definition of runtime parameters (e.g., resource restrictions and affinity settings) for workflow executors and services. The uncertainties in the context come from the execution of the workflows themselves, their interaction with other workflows, and the software/hardware platform. In particular, (1) the changes in the workflows contexts happen in workflows themselves, for instance, a workflow executor fails (e.g., software bug, out-of-memory error), or a workflow service is not available; (2) the changes in the internal system contexts derive from the relationship between the various workflows (and other applications) and how the orchestrator arbitrates their use of the shared platform where they run, for instance, when the resources needed to run a workflow may be in use by other workflows; (3) the changes in the external contexts occur in the underlying platform, including hardware resources, operating systems, and other related systems, which can fail or become unavailable. Those changes can be detected and the manager can react to them but cannot be directly solved at the management level.

B. Multi-layered Control for Autonomic ML Workflows

Each type of uncertainty requires applying different strategies to enable autonomy for ML workflows. These strategies could reside at different layers, for instance, ML workflows could react to changes by restructuring the workflow topology or by reconfiguring the workflow executor/service instances. Thus, we should implement controllers at several layers to manage ML workflows in a completely autonomic way.

A conceptual architecture for multi-layer controlled autonomic ML workflows is proposed in Figure 1, which shows the layers and their interactions. ML workflows are the target system focusing on the ML business. From a static design perspective, ML workflows are composed of some reproducible steps and organized by dependencies. From a dynamic implementation perspective, ML workflows could be run as containerized executors in a pipeline or deployed as online

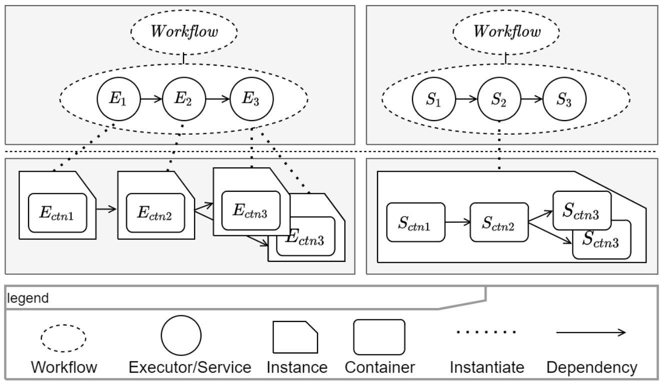

Fig. 2. ML workflows supported by Scanflow-K8s (a batch ML workflow on the left and an online ML workflow on the right).

services consisting of microservice instances. The management system actuates on multiple control layers, namely the application-controlled layer and the infrastructure-controlled layer, which can operate the target system to deal with different types of uncertainty: (1) the application-controlled layer senses the application-related changes, such as the requirements from data scientists and end-users and the workflow context, and restructures the static view of the target system, which is executed with the help from the lower control loops; (2) the infrastructure-controlled layer senses the internal and external system contexts, and adjusts the workflow executors or services at run-time accordingly to the predefined rules of the resource manager by taking advantage of the orchestrator resource management capabilities. In this case, the system autonomy appears as a form of reconfiguration.

C. A Practical Platform for Autonomic ML Workflows

This section describes Scanflow-K8s, a practical platform for autonomic ML workflows that implements the above-mentioned multi-layered control autonomy by means of the integration of a ML workflow manager (i.e., Scanflow) with an orchestrator (i.e., Kubernetes). The whole architecture of this platform is depicted in Figure 3.

Target System: The top of the figure shows the ML workflows which are the target system focusing on the ML business. From a static design perspective, ML workflows define some steps and their dependencies. As shown in Figure 2, two types of workflows are supported by Scanflow-K8s for both training and inference phases. On the left side, a ML workflow is defined as a batch pipeline composed of several executors E_i that are executed in sequence or in parallel. In a batch inference, predictions can be generated asynchronously with a batch of samples and the time to get the results is unconstrained. On the right side, a ML workflow is defined as online services with graph traffic forwarding. In an online inference, predictions are served in real time, typically subject to a latency bound. From a dynamic implementation perspective, the batch executors or the online services are conducted as containerized instances executing locally or in the Cloud. In particular, the executors of batch workflows run

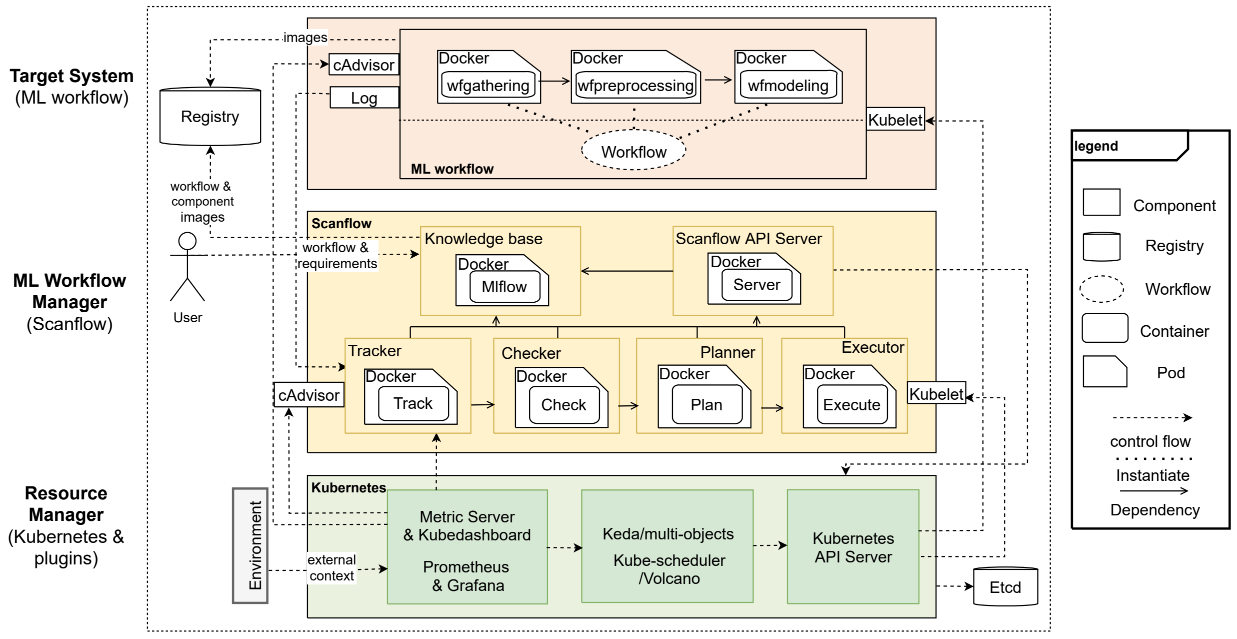

Fig. 3. Scanflow-K8s: A practical platform for autonomic ML workflows.

once for each time the workflow is executed, and the online workflow is deployed as a long-run microservice that is able to deal with client's invocations. Normally, the Data Science team uses the batch ML workflows to build and gain ML models at the ML training phase, while the Data Engineer team conducts the batch ML workflows for batch predictions or deploys online ML workflows in production to make real-time predictions at the ML inference phase.

Application-controlled Layer: ML workflow manager (i.e., Scanflow) is used as a controller of the application layer, as shown in the middle of Figure 3. Scanflow is composed of multiple reactive agents, which work together to perform adjustments in ML workflows to deal with the application-related changes. Internally, Scanflow supports four predefined agent templates, namely tracker, checker, planner, and executor. A tracker-agent, which is based on Mlflow[5], is used to collect the metrics (e.g., number of predictions) or logs (e.g., prediction results) from ML workflows and save information in a knowledge base. A checker-agent can define thresholds to detect outliers or use learning methods to check drift anomalies, which are both based on real-time stream executions and knowledge from a tracker-agent. A planner-agent can decide how to address the detected issues, for instance by retraining the model using transfer learning to improve its robustness based on knowledge from tracker-agent and checker-agent. Finally, the operating plans from a planner-agent can be organized as a set of actions, for example upgrading or changing the version of the model, which are then carried out by an executor-agent, which manages the application-layer internal changes, and the Scanflow API server, which communicates with the infrastructure layer to adjust the target system.

Infrastructure-controlled Layer: The bottom of Figure 3 shows the resource manager working on the infrastructure-controlled layer. Scanflow's best practice is integrating with the well-known Kubernetes orchestrator, given that our ML workflows are wrapped as containers that can be finely managed, and Scanflow-K8s can take advantage from the wide range of toolkits in the Kubernetes ecosystem. Used toolkits are presented in Table II.

TABLE II
KUBERNETES TOOLKITS.

Tool	Role
Kubernetes[6]	Container orchestration, automated container deployment, scaling, and management.
Istio[7]	Service-to-service connection and traffic monitoring.
Prometheus[8]	Metrics monitoring and alerting.
Volcano[9]	Batch workflow scheduling.
Keda[10]	Event-driven autoscaler.
Argo Workflows[11]	Multi-step workflow engine supporting DAG.
Seldon Core[12]	Online model serving on Kubernetes.

The infrastructure-controlled layer supports the deployment and execution of our containerized ML workflows on the platform by leveraging Kubernetes. At the training phase, ML workflows are defined as batch executors and are executed in K8s as Argo Workflows. At the inference phase, ML workflows can be defined both as batch executors or online

[5]https://www.mlflow.org/docs/latest/tracking.html#scenario-4-mlflow-with-remote-tracking-server-backend-and-artifact-stores

[6]https://kubernetes.io/
[7]https://istio.io/
[8]https://prometheus.io/
[9]https://volcano.sh/en/
[10]https://keda.sh/
[11]https://argoproj.github.io/
[12]https://www.seldon.io/

services, according to data engineers' preferences. The former are executed as Argo Workflows (as in the training phase), whereas the latter are deployed and executed using Seldon.

At the infrastructure-controlled layer, the resource manager senses the system and external contexts from the environment, and enables autonomic ML workflows by performing finer-grain adjustments at run-time. For the monitoring, Kubernetes internal metric server and Prometheus toolkit collect the status of the cluster and the performance/resource usage of ML workflows executors or service instances. Also, Istio service mesh traces the traffic and security of each invocation. For the analysis and optimization, the manager can choose the optimal values for the configurable thresholds, which will be used by the HPA (Horizontal Pod Autoscaler) or Keda autoscaler to decide the number of instances, as well as configure the scheduling policy for the default kube-sheduler and the batch scheduler Volcano, which will be used to decide the allocation of ML workflows. Finally, the decided actions are carried out by the Kubernetes API server, which hands out the operations to the kubelet within the cluster to adjust ML workflows in order to adapt to the changing context.

Moreover, the ML workflow manager can govern some changes in collaboration with the resource manager. For example, some application-related run-time information at the infrastructure layer can be tracked by the agents and considered at the application layer. Similarly, some application-related changes in the requirements/decisions need to be implemented in the infrastructure layer, which requires the Scanflow API server to communicate with Kubernetes, for example, to autoconfigure the application thresholds in Keda autoscaler or the affinity/resource limits of workflows according to the user's requirements, and to operate workflows in case of a fail-over to a user-defined backup service.

IV. Agents for Autonomic ML Workflows

In this section, we introduce the architecture of Scanflow agents and their features. In detail, we define the agent communication, triggers, and operation primitives for ML workflows under uncertainties.

A. Agent Architecture

We use the concept of reactive agent, which does not implement a global model or plan but only some simple behaviors. These behaviors allow the agent to react when the environment changes. An agent includes a sensor that senses internal and external state changes, a set of conditional rules that respond to related events, and an actuator that activates a certain process of the environment or other agents.

Scanflow agents are the fundamental components to implement autonomic ML workflows. Each agent is an independent computational unit that is able to run actions according to the state changes. Therefore, an agent can be defined as a set of state-to-action mappings (i.e., $Agent = States(s) \rightarrow Actions(a)$), that is, state changes could result in the execution of actions (if the conditional rules are satisfied). However, an agent usually cannot directly perceive the states but compute

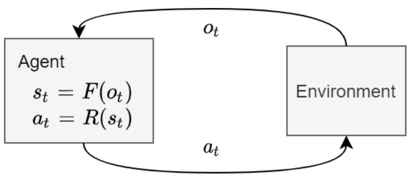

Fig. 4. Agent-Environment interaction.

them from observations o_t using a function F. Also, the agent performs actions through rules with the computed states s_t ($a_t = R(s_t)$). Figure 4 shows the agent-environment interaction: At time t, the agent computes the states s_t from the observations o_t using function F. Then, it chooses actions a_t according to rules R to achieve the agent's goal.

To cope with the autonomic management and supervision for ML workflows, each agent implements its autonomy by defining strategies that include events, constraints, and actions. The autonomic management strategy represents the automation scenarios and can be expressed as 3-tuples $Strategy = (Events, Constraints, Actions)$, where an $Event$ is mainly a state change, which is judged from the observations gathered by the agent triggers, a $Constraint$ is a boolean expression, which refers to whether an attribute value fulfills a condition (e.g., fitting a threshold), and an $Action$ is a single or combined operation primitives or a request to call other agents. Specifically, the autonomic management strategy of the agent is described as: when the $Event$ happens, if the $Constraint$ is satisfied, then the $Action$ will be executed.

B. Agent Social Ability

Social ability describes how multiple agents could collaborate to solve problems by interacting with each other. Traditionally, interaction has been modelled through agent communication languages, such as FIPA-ACL[13]. Recently, researchers have proposed other interaction methods based on concepts like using a shared volume [19]. Scanflow leverages microservice-based agents [23] which could also interact with each other transparently with a service discovery through RESTful APIs. In practice, a single approach of social ability is often insufficient, and thus Scanflow agents apply both shared artifacts and RESTful APIs communication approaches to support social ability of agents.

- Interaction through RESTful APIs: In this approach, the states or actions of an agent are exposed as interfaces. Agents need to be registered first into a service discovery, then they could call the well-defined interfaces from other agents through REST. Normally, the remote call leads to changing the belief/state of the agent and will finally drive an action. Figure 5 exemplifies how Scanflow agents communicate with RESTful APIs. Tracker-agent asks for an agent to check for anomalies in the predicted new data. First, tracker-agent needs to specify which action it wants (e.g., check_predictions); then Scanflow manager

[13]http://www.fipa.org/

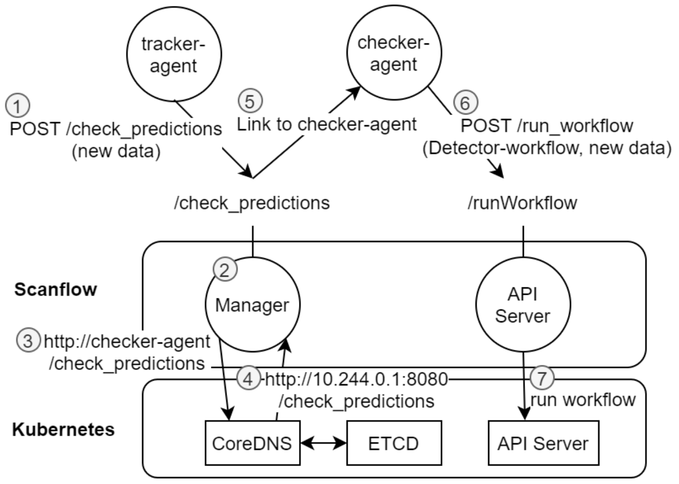
tracker-agent
checker-agent
1 POST /check_predictions (new data)
5 Link to checker-agent
6 POST /run_workflow (Detector-workflow, new data)
/check_predictions
/runWorkflow
Scanflow
2 Manager
API Server
3 http://checker-agent /check_predictions
4 http://10.244.0.1:8080 /check_predictions
7 run workflow
Kubernetes
CoreDNS
ETCD
API Server

Fig. 5. Agents communicate with RESTful APIs.

will generate the service domain name of the agent and request a specific IP address by using CoreDNS, which resolves the domain name, and etcd, which returns the IP address from a service name. Thus, tracker-agent could finally link to the checker-agent. This RESTful POST from the tracker-agent changes the state of the checker-agent, therefore, the checker-agent will POST a run_workflow action to Scanflow API server and Kubernetes API server to carry out its belief (e.g., run detector workflow to check the anomaly of predicted new data).

- Interaction through shared artifacts: This approach communicates through shared artifacts within an application-related knowledge base which receives queries from agents and delivers the results from its database. These include the metadata and logs from the prediction service, and the metrics, scores, parameters, and different versions of the ML model. The states of Scanflow agents can be easily updated through RESTful interaction, but, for complex operations with large data involved, it is more efficient to use shared artifacts so that agents could make actions directly with the accessible resources.

C. Agent Triggers

To actively monitor current $States$, agents are required to trigger tasks to sense the useful observations. Scanflow provides different types of built-in triggers, namely interval triggers, date triggers, and cron triggers (see Table III). Also, the basic triggers can be combined together using ‘and’ or ’or’ logic to produce more complex hybrid triggers. These triggers can be scheduled at a specific time or time intervals to execute tasks so that agents could get required observations to evaluate the changes of $States$. Note that each Scanflow agent contains an asynchronous I/O scheduler with multiple queued tasks. Tasks are run by the scheduler in a thread pool.

On the other hand, an agent can also be triggered by external actions. For example, receiving invocations from other agents, as discussed in Section IV-B.

TABLE III
TYPES OF AGENT TRIGGERS.

Types		Definition
Scheduled Triggers	Interval	Trigger at the specified frequency.
	Date	Trigger once on the given date and time.
	Cron	Trigger when current time matches all specified time constraints (similarly to UNIX cron).
Action Triggers	Call-Receive	Call-Receive interface for agent to be triggered through invocations from other agents.

D. Operation Primitives

After some change in the $States$, the agents need to perform $Actions$ (i.e., $a_t = R(s_t)$). Therefore, we propose some operation primitives that represent the atomic autonomic management steps. The execution of a single primitive or a series of combined primitives is able to implement a full action of an agent. Given that Scanflow agents can manage the ML system by making adjustments both at the application layer and the infrastructure layer (as described in Section III-C), and that both batch and online workflows should be supported, the primitive operations should be designed to happen at those layers and to adapt to those types of workflows.

TABLE IV
AGENT OPERATION PRIMITIVES.

	Application layer	Infrastructure layer

Batch ML workflow	runWorkflow()
	stopWorkflow()
	upgradeWorkflow()
	updateWorkflowAffinity()
	updateWorkflowResource()
		runExecutor()
		stopExecutor()
		upgradeExecutor()
		updateExecutorAffinity()
		updateExecutorResource()

Online ML workflow	deployWorkflow()
	deleteWorkflow()
	upgradeWorkflow()
	updateWorkflowAffinity()
	updateWorkflowResource()
	updateWorkflowReplica()
	updateWorkflowTraffic()
		applyWorkflowInstance()
		deleteWorkflowInstance()
		duplicateWorkflowInstance()

As shown in Table IV, at the application layer, we propose primitives for both types of ML workflows to manage the ML workflow itself, and to set requirements (e.g., affinity, resource limits, etc.) for the workflow from the users’ perspective. At the infrastructure layer, we introduce primitives for operating executors of batch ML workflows or instances of online ML workflows in order to collaborate with the resource manager.

Regarding batch ML workflows, from the application layer, the agents can control the life cycle of the workflow and update its metadata, parameters, and artifacts. For example, the planner-agent can restart the training workflow to re-train the model through $runWorkflow()$; the executor-agent can update the version of the workflow ML model by using $upgradeWorkflow()$ and can update the affinity (using $updateWorkflowAffinity()$) or resource limits requirements (using $updateWorkflowResource()$) with the knowledge from the planner-agent. From the infrastructure layer perspective, the executors will be run and guaranteed by the resource manager, but the agents can actively run or stop an executor using $runExecutor()$ or $stopExecutor()$, respectively. For example, replicated executors can be stopped in case any one of them has finished the task. Also, a specific executor within the workflow can be upgraded, for instance, the planner-agent can update the input parameters for the data-gathering executor of the workflow by using $upgradeExecutor()$ and also change its run-time settings by using $updateExecutorX()$ operations.

Regarding online ML workflows, from the application layer, the agents can add, upgrade, delete, and update the microservice using $deployWorkflow()$, $upgradeWorkflow()$, $deleteWorkflow()$, and $updateWorkflowX()$, respectively. For example, when the model serving service in the workflow needs a new version of the model, the executor-agent must upgrade the microservice by using $upgradeWorkflow()$. The agents can also provide user's requirements to define application-related thresholds. For instance, the planner-agent may call $updateWorkflowReplica()$ to set a failure rate or throughput threshold, so that the online ML workflow microservice will be scaled when the observed value is over the threshold. As for the infrastructure layer, the agents have the option to directly control the number of workflow serving instances. For instance, the executor-agent can call $duplicateWorkflowInstance()$ to scale up and down the online ML workflow service.

V. Case Study And Experimental Analysis

This section presents case studies and conducts experiments on Scanflow-K8s to illustrate the features of the agents and evaluate the feasibility and effectiveness of our agent-based approach for autonomic management of ML workflows.

A. Experimental Setup

Hardware: Our experiments are executed on a ten-node K8s cluster. Each host consists of 2 x Intel 2697v4 CPUs (18 cores each, hyperthreading enabled), 256 GB RAM, 60 TB GPFS file system, and 1-Gigabit Ethernet network.

Software: For all the hosts, we use CentOS release 7.7.1908 with host kernel 3.10.0-1062.el7.x86_64. The Scanflow-K8s platform[14] is built based on Kubernetes v1.19.16 (with Docker 19.03.11, Etcd 3.4.9, Flannel 0.15.0, CNI 0.8.6, and CoreDNS 1.7.0). Its corresponding toolkits (as described in Section III-C) are Istio v1.11.4, Prometheus v14.3.0, Volcano v1.2.0, Keda v2.4.0, Argo Workflows v3.0.0-rc3, and Seldon Core v1.11.2. Additionally, we use Scanflow v0.1.1 with built-in agents for drift detection, which works with MLflow v1.14.1 integrated with a relational database (e.g., PostgreSQL v13.4) for backend entity storage, and an S3 bucket (e.g., Minio Operator v8.0.10) for artifact storage. For the Docker containers used as steps of the ML workflows, Scanflow provides a base executor image using continuumio/miniconda3 and a base service image using python:3.7-slim.

Datasets and Benchmarks: For the first experiment, we use MNIST[15] (60,000 28×28 pixel grayscale images of handwritten digits from 0 to 9) dataset for training a baseline model, and MNIST-C[16] (handwritten digit database with 15 corruptions: corrupted version of MNIST) dataset as new input samples to make predictions. For the second experiment, we use MLPerf Inference benchmark[17] to test batch and online ML inference for image classification, in particular, we use ResNet50 tensorflow model for the ImageNet2012 validation dataset (50,000 images of objects from 1,000 classifications).

To support batch inference in MLPerf, we extended it with the tf2 backend, which supports tensorflow saved_model format, and we packaged both the model and the serving framework in a Docker image, along with a start script to configure MLPerf when launching the container. To support online inference in MLPerf, we extended it with a Seldon backend, so that MLPerf queries can be generated as RESTful invocations and sent to the model serving services. These extensions are available at Github[18].

MLPerf benchmark supports different realistic end-user scenarios through its LoadGen tool. We use the *Offline* scenario, which represents applications where all data is immediately available and latency is unconstrained, to test the throughput (i.e., samples/s) of batch inference workflows, and the *Server* scenario with multiple concurrent LoadGen clients sending queries according to a Poisson distribution to test the throughput (i.e., queries/s) subject to a latency bound (i.e., 6 ms) of online inference workflows.

B. MNIST classification

In this experiment, we show how the various teams will use Scanflow-K8s in the different phases to build and deploy their workflows, as well as the effectiveness of agents that help to manage and supervise the workflows at the application layer while running in production (i.e., to detect and handle drift anomalies). The complete use case is available at Github[19].

1) Various teams build and deploy workflows:

- Training Phase: The Data Science team is responsible for training the ML model to classify MNIST images. Scanflow-K8s supports the definition, building, and execution of batch ML workflows, and runs the various steps of the workflow (i.e. the executors) on Kubernetes by using Argo. Scanflow-K8s allows the modeling step of this workflow to train with different algorithms or with different hyperparameter tuning. Then, the team could select the best model based on the accuracy.
- Inference Phase: After the training, the model is stored in the registry provided by Mlflow and is ready to be used in production. The Data Engineer team should build an inference workflow, so that the trained model can be used to make batch predictions, or deployed as a serving service to allow users to ask for predictions online.

2) Agents implementation: Scanflow agents are responsible for application-layer automation. The four internal supported

[14]https://github.com/bsc-scanflow/scanflow

[15]http://yann.lecun.com/exdb/mnist

[16]https://github.com/google-research/mnist-c

[17]https://github.com/mlcommons/inference

[18]https://github.com/peiniliu/inference/tree/scanflow

[19]https://github.com/bsc-scanflow/scanflow/tree/main/tutorials/mnist

templates of agents are namely tracker-agent, checker-agent, planner-agent, and executor-agent. The Data Engineer team can provide custom functions to enhance the capabilities of each agent. This section evaluates agents which feature a non-trivial data drift detector workflow built from the implementation of the components presented in our previous paper [19]. Checker-agent detects out-of-distribution samples by means of a convolutional deep autoencoder and selects the critical points within these data, which are labelled based on human intervention. Planner-agent leverages transfer learning from the original training workflow to retrain the model after adding the labelled picked critical points to the training data. The autonomic strategies of those agents to manage drift are described in detail in Table V.

TABLE V
AGENTS AUTONOMIC MANAGEMENT STRATEGY

Agent	***Strategies***

Tracker-agent	Strategy: *count_number_of_predictions*
	WHEN $IntervalTrigger(1h, count\_number\_of\_predictions)$
	IF $number\_of\_predictions \geq 1000$
	THEN $Call(Checker\text{-}agent : check\_predictions(newdata))$

Checker-agent	Strategy: *check_predictions*
	WHEN $CallReceive(check\_predictions(newdata))$
	IF $successful\_call$
	THEN $runWorkflow(Detector\text{-}workflow, newdata)$

Planner-agent	Strategy: *retrain_model*
	WHEN $IntervalTrigger(1h, count\_number\_of\_pickeddata)$
	IF $number\_of\_pickeddata \geq 100$
	THEN $runWorkflow(Training\text{-}workflow(production\_model, retrain = True), pickeddata)$
	Strategy: *update_model*
	WHEN $IntervalTrigger(1h, modelaccuracy)$
	IF $newmodelaccuracy > currentmodelaccuracy$
	THEN $Call(Executor\text{-}agent : change\_model(version))$

Executor-agent	Strategy: *change_model_transition*
	WHEN $CallReceive(change\_model(version))$
	IF $successful\_call$
	THEN $updateWorkflow(modelversion, modeltransition)$

3) Application-level autonomy results: Figure 6 presents how a model is autonomously improved by multiple agents in a single interval. At 60 min, tracker-agent sums up the number of predictions during the last one hour (i.e., interval between blue dashed lines: 0-60min). As there are 1000 predictions, checker-agent is triggered to detect the anomalous data (300 anomalous samples are identified) and pick enough new critical data to be appended to the training dataset. As there are 100 new critical samples, planner-agent is triggered to retrain the model and generate a new version. Only those models trained that achieve better accuracy will be iteratively upgraded by executor-agent to be used in production. Figure 7 shows such roadmap of MNIST model upgrades in production. Model V_1 is a baseline model trained by the Data Science team at the training phase with in total 60000 samples and gaining 90% accuracy. The agents monitor predictions over each 1-hour interval (between blue dashed lines) and trigger anomalies detection (between red dashed lines), which might generate a new version of the model for each interval. From those upgraded models, over time only V_2, V_3, and V_7 have been used for predictions in production because they provided better accuracy than the former ones (e.g., V_2: 91%, V_3: 92%, and V_7: 93%). This demonstrates that Scanflow agents can provide autonomy at the application level to help ML workflows to maintain the model accuracy when facing constantly evolving data profiles.

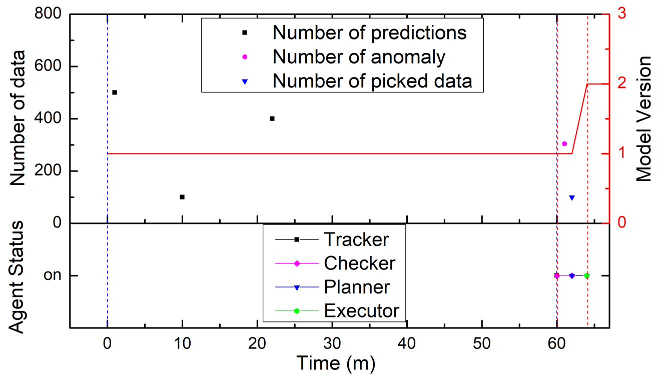

Fig. 6. Agent-based model debugging in the presence of data drift.

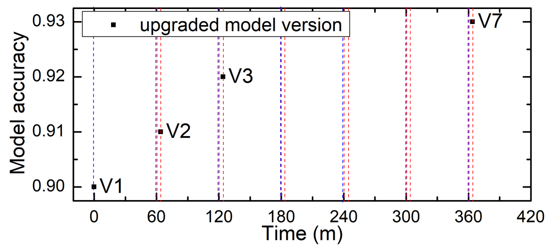

Fig. 7. Road map of MNIST model upgrades in production.

C. MLPerf Inference Benchmark

In this experiment, we show how Scanflow-K8s can deal with both context changes and non-functional requirements by taking advantage of the resource manager and also the collaboration between application and infrastructure layers. This use case is also available at Github[20].

1) Automation at the infrastructure layer: Automation at the infrastructure layer allows taking advantage of the resource management capabilities of the orchestrator to improve the reliability, scalability, and load balancing of workflows. The infrastructure layer provides simple strategies to deal with some system contexts such as self-healing, auto-scaling based on observed system metrics such as CPU utilization, and load-balancing in a round-robin option [24]. However, as they use low-level system information, these strategies are less expressive and more difficult to configure for the end-user, as demonstrated in the next section.

2) Multi-layered Control for Autonomic ML workflows: This section shows the benefit of considering application-provided knowledge to perform resource management actions.

First, we compare infrastructure- vs. application-level autoscaling by using 100 LoadGen users asking for predictions

[20]https://github.com/bsc-scanflow/scanflow/tree/main/tutorials/mlperf

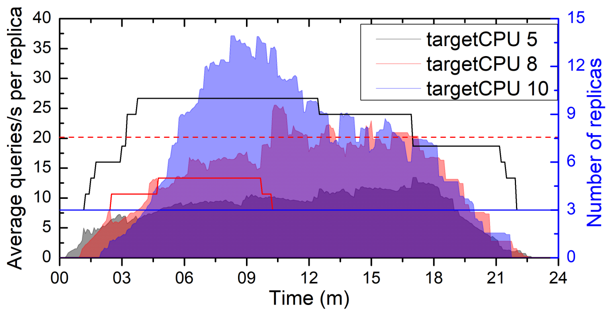

Fig. 8. Auto-scaling driven by CPU utilization metric.

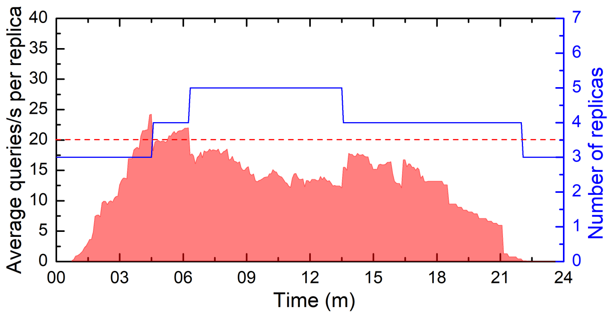

Fig. 9. Agent-tuned auto-scaling driven by application-level metric.

in the *Server* scenario while expecting a given service QoS (e.g., average queries/s per replica < 20). In Figure 8, the data-engineer uses different infrastructure-related settings to define the auto-scaling threshold (i.e., setting the target CPU utilization to 5, 8, or 10 CPUs). The workflow is rapidly scaled up (i.e., number of replicas is increased) at the beginning when setting CPU utilization threshold to 5, hence the system wastes many resources to fulfill the throughput requirement. The workflow is never scaled when setting CPU utilization threshold to 10, thus does not mostly satisfy the throughput requirement. Setting CPU threshold to 8 mitigates the problems of the other two settings, but it is still not matching exactly the QoS requirement. This shows how hard is for the data-engineer to find the optimal auto-scaling settings when using only infrastructure-related metrics. Figure 9 shows agent-tuned auto-scaling according to an application-level non-functional QoS requirement provided by the end-user. The planner-agent can autonomically replace the data-engineer to tune the auto-scaling threshold of Keda to meet the requirement. The workflow is scaled up when the real-time throughput goes over the threshold, and scaled down when facing a low load. That is to say, having the application-layer knowledge allows the agents to manage resources wisely by matching the threshold with the QoS requirement in the service level agreement.

At this point, we evaluate the agent-tuned anti-affinity for batch workflows, which allows to constrain which nodes they are not eligible to be scheduled based on the pods that are already running on the nodes. We use 50 LoadGen users in the *Server* scenario to stress out an online inference service, while in the meantime another LoadGen user asks for a batch

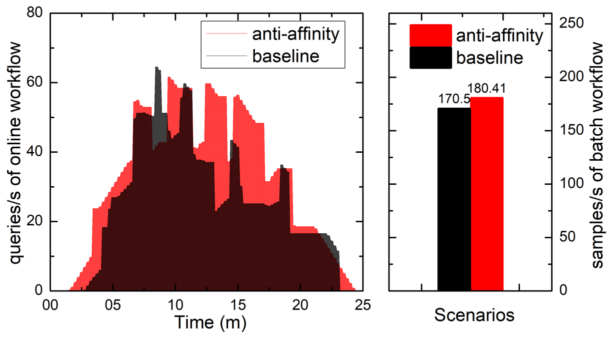

Fig. 10. Agent-tuned anti-affinity.

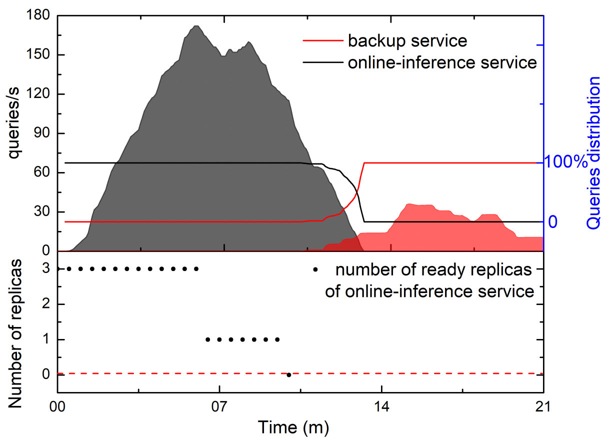

Fig. 11. Agent-tuned service fail-over and traffic redirection.

prediction by means of the *Offline* scenario. In the baseline configuration, the batch and online inference workflows are colocated in the same node; while in the anti-affinity configuration, the planner-agent sets anti-affinity of the batch vs. the online workflow, so that they are allocated separately. Figure 10 shows the benefit on the performance of both workflows when agents define their anti-affinity, because each workflow can use the spare resources in its allocated node, which would be otherwise used by the colocated workflow if they are executed together, as shown in the baseline.

Finally, we demonstrate how Scanflow-K8s can deal with workflow internal faults by means of replica fail-over driven by application-level information. In particular, if the inference service is not available and it cannot be recovered by restarting the service instances at the infrastructure layer, a backup service deployed at the initiative of the Data Engineer team can take over and Scanflow-K8s redirects all the traffic from the original inference service to the backup service to maintain the availability. We show the queries distribution between these two services in Figure 11. We have started 200 LoadGen users in the *Server* scenario so that replicas of the original inference service start to fail the readiness health-check due to the high load. When the planner-agent detects that the online-inference service is not available (i.e., its number of ready replicas is 0), it dynamically redirects the query traffic from the unavailable service to the backup service. This is possible thanks to the application knowledge that both services are equivalent, since from the infrastructure perspective they are different services.

The above experiments exemplify how the agents can leverage application-layer knowledge to enhance resource management actions. In the first one, the agent used arbitrary application-level metrics to configure auto-scaling according to QoS requirements. In the second one, the agent tuned the container-level resource and affinity configuration to optimize performance according to workflow type and resource availability. In the last one, the agent dealt with service unavailability by redirecting the traffic to a backup service defined at the application level. Application-layer knowledge is currently provided by the end-user/data-engineer, but the agent strategies could be enhanced to gather knowledge from other sources (e.g., other models, expert knowledge base).

VI. Conclusions And Future Work

This paper presented Scanflow-K8s, an agent-based framework that enables autonomic management and supervision of the end-to-end life-cycle of ML workflows at Kubernetes clusters. We evaluated two simple use cases, although we engineered the framework so that it can be easily adapted to different ML workloads and more complex adaptation scenarios. First, we used a MNIST project to show how different teams could leverage Scanflow-K8s to manage ML workflows at different phases and how its agents collaborate to debug a drift anomaly problem and upgrade a new model. Second, we used ImageNet2012 classification from MLPerf benchmark for batch and online inference scenarios to show how agents take actions to keep the performance and availability of workflows in this multi-layer controlled autonomic architecture. We provided some template agents to be used in these use cases, but as future work, we plan to implement more generic template strategies and user interfaces so that developers could easily bring their knowledge or the insights learned from other models to the agents. We will also develop more complex (and more dynamic) adaptation policies both at the application and the infrastructure layers, and the needed enhancements in the framework to enforce them at scale (management of conflicts among multiple strategies, agent throughput under high load, etc.).

Acknowledgment

This work was supported by Lenovo as part of Lenovo-BSC 2020 collaboration agreement, by the Spanish Government under contract PID2019-107255GB-C22, and by the Generalitat de Catalunya under contract 2017-SGR-1414 and under grant 2020 FI-B 00257.

References

[1] Google Cloud, "Machine learning workflow," 2021. [Online]. Available: https://cloud.google.com/ai-platform/docs/ml-solutions-overview

[2] Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, and Y. Yu, "Taking Human out of Learning Applications: A Survey on Automated Machine Learning," 2019. [Online]. Available: https://arxiv.org/abs/1810.13306

[3] Run.AI, "Machine Learning Workflow: Automating Machine Learning Workflows," 2021. [Online]. Available: https://www.run.ai/guides/machine-learning-engineering/machine-learning-workflow/#Automating

[4] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren, "Aslib: A benchmark library for algorithm selection," *Artificial Intelligence*, vol. 237, pp. 41–58, 2016.

[5] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and Robust Automated Machine Learning," in *Proc. of the 28th Intl. Conference on Neural Information Processing Systems - Vol. 2*, ser. NIPS'15. MIT Press, 2015, pp. 2755–2763.

[6] G. Katz, E. C. R. Shin, and D. Song, "ExploreKit: Automatic Feature Generation and Selection," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016, pp. 979–984.

[7] Google Cloud, "MLOps: Continuous delivery and automation pipelines in machine learning," 2021. [Online]. Available: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[8] J. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, Mar. 2014.

[9] 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.

[10] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "TensorFlow-Serving: Flexible, High-Performance ML Serving," 2017. [Online]. Available: https://arxiv.org/abs/1712.06139

[11] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: Machine Learning Inference Serving on Serverless Platforms with Adaptive Batching," in *Proc. of the Intl. Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC'20. IEEE Press, 2020.

[12] C. Cox, D. Sun, E. Tarn, A. Singh, R. Kelkar, and D. Goodwin, "Serverless inferencing on Kubernetes," 2020. [Online]. Available: https://arxiv.org/abs/2007.07366

[13] D. J. Kedziora, K. Musial, and B. Gabrys, "AutonoML: Towards an Integrated Framework for Autonomous Machine Learning," 2020. [Online]. Available: https://arxiv.org/abs/2012.12600

[14] P. Liu, X. Mao, S. Zhang, and F. Hou, "Towards reference architecture for a multi-layer controlled self-adaptive microservice system," in *Proceedings of the 30th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2018, pp. 236–241.

[15] T. De Wolf and T. Holvoet, "Towards autonomic computing: agent-based modelling, dynamical systems analysis, and decentralised control," in *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN)*, 2003, pp. 470–479.

[16] G. Tesauro, D. Chess, W. Walsh, R. Das, A. Segal, I. Whalley, J. Kephart, and S. White, "A multi-agent systems approach to autonomic computing," in *Proc. of the 3rd Intl. Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2004, pp. 464–471.

[17] F. M. Brazier, J. O. Kephart, H. V. D. Parunak, and M. N. Huhns, "Agents and Service-Oriented Computing for Autonomic Computing: A Research Agenda," *IEEE Internet Computing*, vol. 13, no. 3, pp. 82–87, 2009.

[18] I. Zliobaite, A. Bifet, M. Gaber, B. Gabrys, J. Gama, L. Minku, and K. Musial, "Next Challenges for Adaptive Learning Systems," *SIGKDD Explor. Newsl.*, vol. 14, no. 1, pp. 48–55, Dec. 2012.

[19] G. Bravo-Rocca, P. Liu, J. Guitart, A. Dholakia, D. Ellison, J. Falkanger, and M. Hodak, "Scanflow: A multi-graph framework for machine learning workflow management, supervision, and debugging," 2021. [Online]. Available: https://arxiv.org/abs/2111.03003

[20] J. Klaise, A. V. Looveren, C. Cox, G. Vacanti, and A. Coca, "Monitoring and explainability of models in production," 2020. [Online]. Available: https://arxiv.org/abs/2007.06299

[21] L. Wang, L. Yang, Y. Yu, W. Wang, B. Li, X. Sun, J. He, and L. Zhang, "Morphling: Fast, Near-Optimal Auto-Configuration for Cloud-Native Model Serving," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC'21. ACM, 2021, pp. 639–653.

[22] A. Imbrea, "An empirical comparison of automated machine learning techniques for data streams," Ph.D. dissertation, January 2020. [Online]. Available: http://essay.utwente.nl/80548/

[23] R. W. Collier, E. O'Neill, D. Lillis, and G. O'Hare, "MAMS: Multi-Agent MicroServices," in *Proceedings of the 2019 World Wide Web Conference*, ser. WWW'19. ACM, 2019, pp. 655–662.

[24] Kubernetes, "Why you need Kubernetes and what it can do," 2021. [Online]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/#why-you-need-kubernetes-and-what-can-it-do