

D5.5 Specification of AI methods for FRACTAL system control

Deliverable Id:	D5.5
Deliverable Name:	Specification of AI methods for FRACTAL system control
Status:	V0.3
Dissemination Level:	Public
Due date of deliverable:	M26 October 2022
Actual submission date:	10/30/2022
Work Package:	WP5 "AI and Safe Autonomous Decisions"
Organization name of lead contractor for this deliverable:	Zylk.net
Author(s):	Alfonso González, ZYLK Andrea López, ZYLK Sergio Martín, ZYLK Andoni Angulo, ZYLK Vahid Mohsseni, UOULU Huong Nguyen, UOULU Abhishek Kumar, UOULU Lauri Loven, UOULU Alexander Flick, PLC2 Ignacio Garrido, IFT
Partner(s) contributing:	ZYLK UOULU PLC2 PROINTEC

Abstract:

This deliverable is a part of FRACTAL WP5 Task 5.4, and details the results of the research done on AI methods for FRACTAL System Control. It dives into the concept of MLOps, a set of methodologies to optimize AI processes, Orchestration strategies at its various levels of application, and MLBuffet, an ML model server for model, training and deployment developed in T5.4



This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056



Co-funded by the Horizon 2020 Programme of the European Union under grant agreement No 877056.



Project	FRACTAL
Title	Specification of AI methods for FRACTAL system control
Del. Code	D5.5

Contents

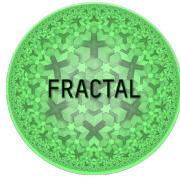
1 History.....	3
2 Summary.....	4
3 Introduction.....	5
3.1 Objectives and Approaches.....	5
4 MLOps.....	6
4.1 From DevOps to MLOps.....	6
4.2 Machine Learning Life-Cycle.....	6
4.3 Tools.....	7
4.4 MLOps Pipeline.....	11
5 Applied Orchestration.....	13
5.1 Orchestration in FRACTAL nodes.....	13
5.2 Top-down orchestration.....	13
5.3 Bottom-up orchestration.....	14
5.4 Container orchestration.....	15
5.4.1 Docker Swarm.....	17
5.4.2 Kubernetes.....	19
5.5 Self-orchestrating systems.....	20
6 MLBuffet v2.....	21
6.1 Introduction.....	21
6.2 Implementation.....	21
6.3 How does it work.....	23
6.4 Installation.....	24
6.5 Other implementations.....	25
6.5.1 ARM architecture (VERSAL node).....	25
7 Conclusions.....	26
8 List of figures.....	27
9 List of tables.....	28
10 List of Abbreviations.....	29



Project	FRACTAL
Title	Specification of AI methods for FRACTAL system control
Del. Code	D5.5

1 History

Version	Date	Modification reason	Modified by
V0.1	06/08/2022	Initial document and content proposal	Alfonso González, ZYLK
V0.2	08/09/2022	First content update	Alfonso González, ZYLK
V0.3	30/09/2022	Final content update and polishing	Alfonso González, ZYLK
V1.0	26/10/22	Final version after internal reviews	Alfonso González, ZYLK



Project	FRACTAL
Title	Specification of AI methods for FRACTAL system control
Del. Code	D5.5

2 Summary

This deliverable covers the main research outcomes from T5.4, introducing the concepts of Machine Learning Operations (MLOps), a subset of Development Operations (DevOps) methodologies, Orchestration, providing alternative strategies to orchestrate and manage system and service automation (top-down, bottom-up and self-orchestrating approaches), and describing the implementation result of the research and developments on these topics, the MLBuffet tool to orchestrate the Fractal control system based on Artificial Intelligence (AI) approaches.

There are three main sections (4, 5 and 6) in D5.5:

Section 4 explains what the MLOps methodologies are, thoroughly explaining the necessity to apply DevOps methodologies to Machine Learning (ML) projects, and giving all the insights and information for the Use Cases (UCs) to implement the set of tools and frameworks available to perform MLOps strategies into their working pipelines, providing practical implementations and example pipelines to cover all the steps in a typical ML process.

Section 5 gathers all the required information about Orchestration in ML systems, explaining the different orchestration strategies from a theoretical perspective (focusing on top-down and bottom-up orchestration), and how these can be applied into physical systems to conform a Fractal node architecture. This section is of special importance because it allows WP6 with a practical implementation orchestration framework, and is the converging spot for the ML Orchestration (WP5) and Systems and Service Orchestration (main focus of WP6) developments. The theoretical aspects of orchestration as studied in T5.1 are particularized here for Fractal systems and an overview of the applicability of these strategies to Edge nodes is provided, introducing containerization and container orchestrators.

Finally, Section 6 details the main outcome of the developments done throughout the task, the MLBuffet tool. This open-source implementation of an Edge ML server for inference, training, and storage of models has been fully developed in the Fractal project framework, specifically in T5.4. The main feature of this tool is that it enables the orchestration mechanisms for ML models in the Edge, so that orchestration can be performed over the ML artifacts in terms of storage, training and inference. This is done in a virtualized environment that ensures universality and automation of the deployments.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

3 Introduction

The objective of this deliverable is to show the results of the research that has been performed during T5.4 in WP5, which was focused on building an AI-based FRACTAL system control. The starting point of this task was the theoretical research being performed in parallel to T5.1, where the theoretical framework for a Fractal platform and Fractal AI was being done. From there, an analysis of the existing AI and orchestration tools was done, and the existing methodologies for continuous integration and continuous development (CI/CD) for Machine Learning (ML) processes were researched. This leads to the concepts of MLOps and Orchestration, which later enabled the possibility of building a new open-source tool that can give answer to the Fractal Edge system orchestration necessities.

3.1 Objectives and Approaches

The research of this task was done following the overall objectives of the FRACTAL project, gathered in the Table below:

Objective 1	Design and Implement an Open-Safe-Reliable Platform to Build Cognitive Nodes of Variable Complexity
Objective 2	Guarantee FRACTAL nodes and systems extra-functional properties (dependability, security, timeliness and energy-efficiency)
Objective 3	Evaluate and validate the analytics approach by means of AI to help the identification of the largest set of working conditions still preserving safety and security operational behaviors.
Objective 4	To integrate fractal communication properties (scale free networks) to FRACTAL nodes.

Table 1 - FRACTAL Project objectives

Specific objectives of this task are based on WP5 objectives:

- Study and enhancement of AI methods on the FRACTAL edge computing architecture.
- This WP will concentrate on particular set of AI methods to study further and implement on the platform, based on the theoretical results and architecture of T5.1 as well as the platform of T5.2.
- Enhance the internal operation of the FRACTAL system and enable new functionalities for prediction, system orchestration, and autonomous control of the system nodes

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

4 MLOps

4.1 From DevOps to MLOps

The MLOps concept cannot be understood without the DevOps approach. The DevOps term comes from ‘Development and Operations’, two of the traditional groups of workers from the IT companies.

In order to speed up and facilitate the development process and the submission of new software and its updates, teams that usually do not work together, such as the development and maintenance groups, join forces and contribute to the software development process. The goal of that collaboration is to optimize the developers' workload while ensuring the functionality of the generated applications with the monitoring that the operations team can provide.

As IT companies started to integrate teams specialized in Machine Learning into their staff, new requirements that were not in the scope of DevOps practices and tools arose. To satisfy them, MLOps concept was developed, which combines DevOps with Machine Learning management tools and practices. As in DevOps, the term MLOps comes from the union of Machine Learning and Operations.

The application of MLOps practices accelerates the processes of experimentation and model development, making the deployment and maintenance of the models in production more efficient, ensuring the quality of the results. This new approach joins the DevOps teams, machine learning research teams, engineers, and data analysts who are in charge of the design and maintenance of models and the preparation of data for continuously training and fine-tuning the models.

4.2 Machine Learning Life-Cycle

Similar to DevOps practices and software development stages, MLOps cover each step involved in Machine Learning model development. Those steps can be divided into three main phases: experimentation, production, and monitoring.

The goal of the first phase is to design and develop the machine learning model. First, problem identification and data collection, analysis, and labeling are a must to design a model that can comply with the requirements established by the client. Along with that, a model selection process is conducted by studying different types of models to find the one that best fits the requirements of the Use Case. Lastly, the first experiments are carried out to obtain a preliminary model that will be later fine-tuned.

When a machine learning model that works as desired has been developed, the next step is to put that model into production and test its capabilities with real-world data; that is, data associated with the problem that had to be solved in the first place. During this phase, different experiments and training stages will be performed to fine-tune the hyperparameters of the model, and each version of the

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

model is saved and stored following data version control rules so that every experiment is backed up correctly.

Finally, once the model has been served and put into production, a set of processes will be carried out to ensure that the deployed machine learning application's performance is always optimal. This requires, for example, monitorization of metrics such as the accuracy and precision of the model when encountering and processing new data, in order to detect any signs of model decayment or data drift as soon as possible, and fix those issues by retraining and tuning the model when necessary.

4.3 Tools

Different tools and software have emerged recently with the purpose of covering each of the steps mentioned in the previous section. While some of the software is more specialized in one specific task such as data management or version control, other tools have been designed with the aim of performing tasks related to more than one stage of the MLOps life cycle.


Starting with the software packages that cover more than one stage of the lifecycle, **MLflow** is one worth mentioning. It is an open-source platform designed to manage the entire MLOps lifecycle, performing functions related to experimentation, reproducibility, and deployment, including a central model registry. Also, one noticeable feature of this tool is that it is compatible with any machine-learning library and programming language.

In order to organize and manage the machine learning models, MLflow is comprised of four main components:

- (i) a tracking system, that has the purpose of tracking and recording different experiments, allowing the data scientist to compare the different results;
- (ii) the models component, managing and deploying the models from different machine learning libraries, allowing to serve them in different inference platforms;
- (iii) the projects, that package the code developed for the machine learning model in a reusable and reproducible way;
- and finally (iv) the model registry, that manages each step of the model lifecycle in a centralized fashion.

Along with those features, MLflow also offers the possibility of hosting the MLflow models as REST endpoints.

Kubeflow is another tool capable of handling multiple steps of the MLOps lifecycle. This software is an open-source machine learning toolkit that works on top of Kubernetes, taking advantage of the functionalities that this platform provides intending to make the deployment, experimentation, and model serving phases simpler. Kubeflow enables different users to build their ML experiments independently within the current cloud scope. It is a beneficiary approach since it utilizes cloud computing resources, resulting in total cost reduction. Kubeflow includes services to create and manage interactive notebooks such as Jupyter, allowing the scientist to experiment with local workflows before deploying them to the cloud. With that, Kubeflow works with pipelines that help manage and deploy

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

the ML workflows from end-to-end, making it possible to schedule different runs of an experiment and compare the results which can be used in parameter tuning. KubeFlow also includes support for different training setups, supporting GPU-driven training when available, and allows to export and serve the trained Tensorflow models using a Tensorflow Serving container.

Focusing on more specific tools, **Apache Airflow** is an open-source platform that has the aim of monitoring, scheduling, and managing workflows. Even though it is not designed specifically for machine learning models and MLOps lifecycle, this tool can be useful for planning small tasks related to the whole lifecycle. Allows the user to plan different workflows by using cron expressions and directed acyclic graphs that can be programmed using Python language, where the dependencies among the tasks can be specified, and provides a web graphical user interface to check the status of the scheduled pipelines and the tasks that comprise them.

Following with data versioning and version control, **DVC** is by far one of the most popular tools. This is an open-source version control system designed for coping with versioning and organization of data inside the machine learning environment. It is storage agnostic, and the data management system that DVC implements keeps track of the evolution of every machine learning model under development. In addition to these features, DVC stores the code and data needed in each experiment in a consistent way, making every experiment reproducible.

Finally, data metrics and visualization tools worth mentioning are **Grafana** and **Prometheus**. These two tools are open-source software packages with different purposes: on the one hand, Prometheus is a software designed to collect metrics by scraping different endpoints and storing them as time series, allowing the user to monitor systems by defining different alert rules for the stored metrics. On the other hand, Grafana is a tool designed to create interactive and highly customizable data visualization dashboards of real-time data, that can be collected from various data sources specified by the user. Combining both tools, it is possible to create a monitoring system that displays the state of a Machine Learning model in real-time, allowing the user to schedule different retraining sessions when needed, or notifying when the performance of the model starts decaying by setting the corresponding alert rules.

Labeling tools in supervised learning

One of the first steps in any project that includes a supervised artificial intelligence component, is to obtain a good dataset with which the model is trained. Although there is an enormous variety of datasets available on the Internet, in many cases it is necessary to manually label the dataset. This can be either because there is no data for the faced problem, or to better adapt the models to the specific problem by adding a fine-tuning phase. Moreover, the success of any of these projects is closely related to the quality of the dataset used.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

Name	Language	Data Type	Source Code
CVAT	Typescript, React, CSS, Python	Image	https://github.com/opencv/cvat
awesome-data-labeling	Python	Image, audio, text, time series	https://github.com/heartexlabs/awesome-data-labeling
bbox-visualizer	Python, makefile	Image	https://github.com/shoumikchow/bbox-visualizer
dataqa	Python	Text	https://github.com/dataqa/dataqa-python
doccano	Python	Text, sequence	https://github.com/doccano/doccano
hover	Python	Image	https://github.com/phurwicz/hover
Label-studio	Python	Image, audio, text, time series	https://github.com/heartexlabs/label-studio
Labelme	Javascript	Image	https://github.com/wkentaro/labelme
VoTT	Typescript	Image	https://github.com/microsoft/VoTT
Yolo-mark	-	Image	https://github.com/AlexeyAB/Yolo_mark

Table 2 - List of available data labeling tools

Having a dataset with good labels is as important as the model itself. For this reason, the labeling stage is a fundamental part of any artificial intelligence project and must be carefully planned.

There are multitude of open-source annotation tools available. Some of the most widely used are **LabelMe**, **LabelStudio** and **CVAT**. Labelme provides an online tool to do image labeling. LabelStudio can be installed locally and it allows to configure the annotation interface with a configuration JSON. Furthermore, LabelStudio specializes in both image labeling and natural language processing (NLP). Finally, CVAT can be either installed locally or accessed online, and it is particularly suitable for video labeling.

For example, for crack labeling, after selecting the appropriate images from all those obtained, the "Labelme" program is used to generate the masks.


	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5



Figure 1: Valid photo examples

Manual labeling work consists of drawing a polygon that runs along the contour of the detected pathology.

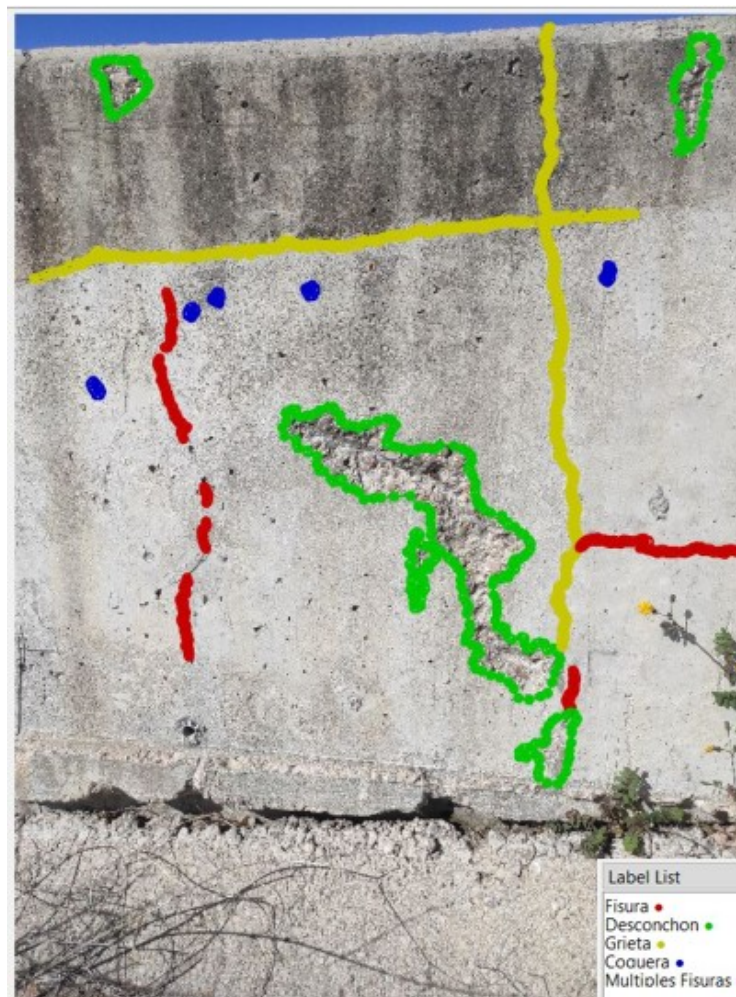


Figure 2: Example of labeling different types of defects

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

As seen in the image above, labeling can be done not only with fissures or cracks but also with other surface defects or irregularities. In this way the model can learn to differentiate some defects from others (paint defects, gravel nests due to lack of vibration during execution, concrete joints, etc...).

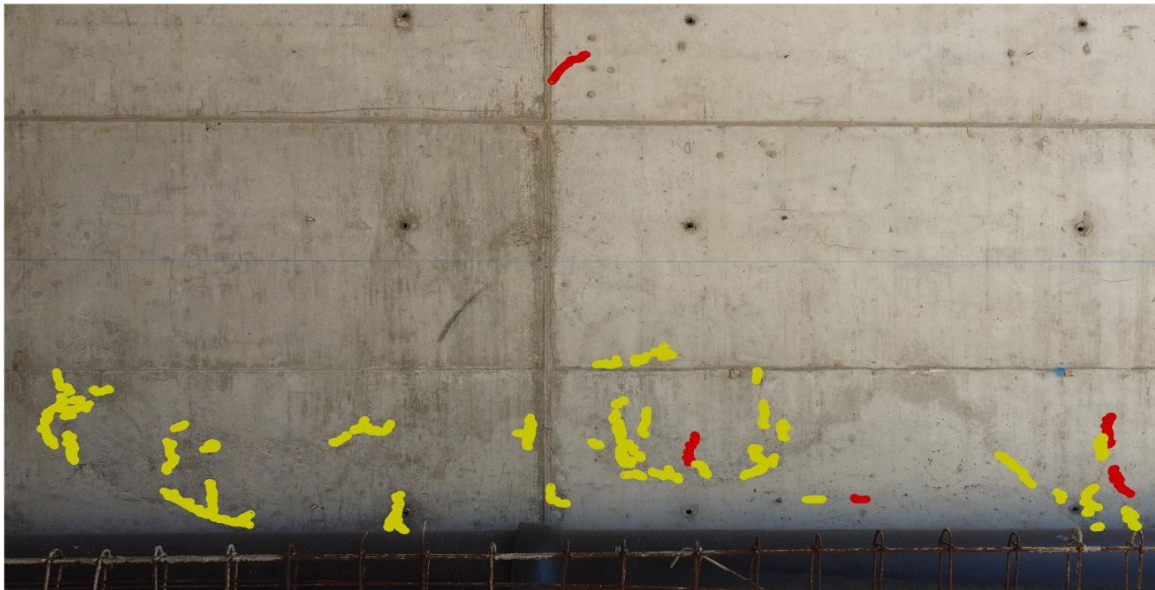


Figure 3: Example of labeling considering only cracks and fissures

Another way of labeling that has been used is to mark only the cracks or fissures so that the model learns to distinguish them from everything that is not (see Figure 3).

4.4 MLOps Pipeline

The main purpose of the MLOps practices is the management of the life-cycle of the models, from code and data versioning to the deployment, and covering intermediate steps such as model building and versioning.

An example of this is depicted in Figure 4

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

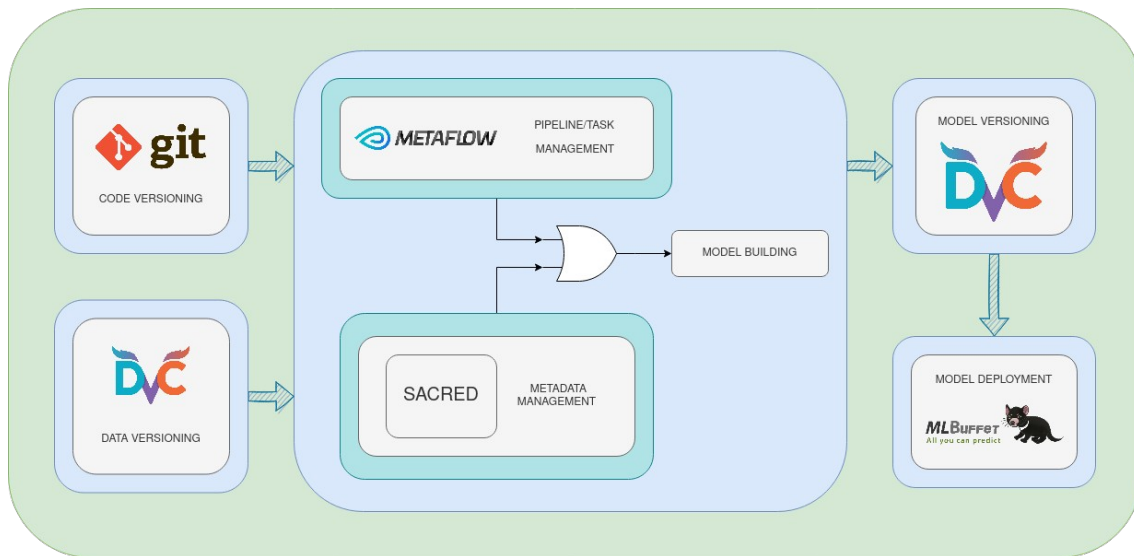


Figure 4: Example of a MLOps pipeline

- The code containing the definition of the model is versioned into a repository, following the GitOps strategy. These tools ensure that the code is maintained and updated.
- The data versioning is essential to train the model with the most up to date dataset. The dataset can be grown up with new data, and the new versions of the model should be trained with that dataset. One of the most used tools for that feature is DVC, which is transparent with the different types of train dataset, such as images, CSV, texts, etc.
- The pipeline/task management is a feature of the MLOps approach where the model's code is managed. The use of a tool governing the pipeline is essential to ensure the correct behavior of each step of the model training. Metaflow is shown in this example, which has the role of dividing the model building into steps, ensuring the correct path of the training and, in addition, with the capability to parallelize some steps if necessary.
- When the model is built, every new version developed should be saved and distinguished from the other versions, allowing access to all of them. DVC complies with this feature acting like it does with the data versioning.
- The deployment of the model is the final target of the lifecycle of the model. An effective deployment is essential to ensure a correct inference with it. For that reason, under the Fractal project MLBuffer has been developed to deploy models via HTTP requests on Kubernetes.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

5 Applied Orchestration

5.1 Orchestration in FRACTAL nodes

The focus of a FRACTAL system to support a high number of heterogeneous edge nodes brings forward the need for efficient management, allocation and distribution of resources as promoted in D5.2. The orchestration concepts introduced there describe the device-edge-cloud computing continuum to summarize all potential resources. The specific elements within the continuum range from energy, direct access hardware elements, virtualized resources, workflow pipelines and applications, where each node may be providing any subset thereof.

To properly allow for seamless computation spanning the full set of physical and virtual nodes determines the orchestration strategy and complexity required to achieve best possible efficiency. As various distinct orchestration targets can be identified, like topology, network or services, each brings a potentially different strategy. To enable this for larger or complex systems the orchestration can resort to utilize edge AI deriving the cooperation steps with local intelligence based on monitoring. Such approaches are derived in WP4 of the FRACTAL project and are enabled by the growing computation power of edge node platforms as chosen for this project.

In the sections to follow the mission mode or task orchestration will be considered first place to explain the principles of centralized orchestration strategies in contrast to decentralized approaches. Such choice for a particular task-level orchestration would be supported differently based on the node scaling in the system and therefore is taken differently in particular use cases. In further sections the orchestration on application level, or service level respectively, through the life-cycle control of containerized services Kubernetes is derived.

5.2 Top-down orchestration

Top-down orchestration requires a centralized control system to manage the resources and distribute the tasks. One resembling model of the top-down approach is the process scheduler in operating systems. There should be a stateful mechanism to keep track of every resource status in the design and then can dispatch the tasks through the nodes to run. This approach has some challenges, including but not limited to:

- **Statefulness.** The control unit should be aware of any condition and status of the resources in the system for the correct resource allocation.
- **Efficient algorithm.** Selecting the algorithm for the scheduler is an essential part of the orchestration in cases such as minimizing the queue time, full utilization of the resources, etc.
- **Fault tolerance.** The failure of the centralized dispatcher means the failure of the system without the recovery plans.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

Although the challenges mentioned earlier exist in the top-down approach, it has a generalized method to be beneficiary in situations where the computing continuum is desired. This is due to the nature of this approach, which is having a global view over the resources at the cost of autonomy of slave nodes. Here, the computing continuum may refer to combining the cloud, edge, fog, and even low-end devices with the ability to communicate data in between. The meta-information, such as the geolocation of the nodes, the computing capacity, and the availability of the power about the nodes, gives the control plane the insight for better task distribution.

5.3 Bottom-up orchestration

The bottom-up is another strategy to dispatch the workload to different edge devices to achieve load balancing. It comes when considering some of the limitations of top-down approaches with more flexibility and democracy. As its name, everything starts from the bottom, the local nodes in this context. The flexibility and democracy here are expressed via the willingness of taking the task of each node, therefore, nodes can freely choose the task they want to process based on their current resource status. This also means that there is no central control unit and no forcing from it to make you take the task in any way as it would be in the top-down approach.

With this strategy, the local nodes will make communications with each other to make an agreement about the task they want to pass or exchange. Basically, the receiving node will self-evaluate if it can handle the task or not (based on its current resource) and reply: yes or no for the asking. No one will know and care about the others' resource information, they just need the acceptance from the receiving node and make an agreement via communication here.

Assuming that we have 3 local nodes (A, B, C) in the system, then an example of the bottom-up approach in task orchestration could be:

- One task was initially assigned to A (At this time, A is overloaded and wants to pass the task, so A needs to ask around)
- A started a conversation with B and asked for help. However, at this time, B was also overloaded and B said NO
- A cannot force B to do once B says NO, then it needs to keep asking around for help, so A asks C
- C was free at that time and C said YES
- A transferred the task to C

As a notable point, the proper way to evaluate on the node itself to see if it has sufficient power to process the task is still an open question without the best answer. Workload and computing power assessment methods of one device are many and varied. The possible solutions can be addressed as follows:

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

- based on the average number of tasks completed per unit of time
- simply the amount of free resources at the evaluation time
- or a combination of both.

Even though this democratic approach makes harmony and happiness among nodes when exchanging the task, it also contains more risks when it cannot minimize the delay time. Related to this, the main reason is that it does not consider the estimated time processing at the receiving node and the two-way communication latency. Therefore, if a task comes in a long queue of one node that has a current heavier workload than others, it still can be solved even faster (sequentially) than when it is transmitted to another far free node, waiting for the processing and transmitted back for the result. From that, we can see the biggest cons of this approach is just considering different factors (e.g. resources, communication, energy, etc.,) in isolation while there are implicit interactions between them. Particularly, computational time and the energy consumption to make communications among devices are one of the most common combinations that we need to concern about when negotiating to reach a consensus on an optimal task allocation.

Beyond the foregoing, this approach can also be interpreted in the way that the lower-layer devices (e.g. sensors or data collectors) try to process things first (on-board processing) before it gets overloaded and needed to send the workload to the higher-layer devices such as fog, edge devices or even cloud server, where has more computing power to continue the processing progress. This way releases the congested work that needs to be processed on the server and the near-data processing would also minimize a lot of latency.

5.4 Container orchestration

Not only nodes and physical systems are entities subjected to be orchestrated. Virtual nodes, virtual machines, and containers (which are a way of virtualization) can also be orchestrated with the various strategies described above.

Containers are a particular entity when referring to virtualized environments, because they are packages of software and processes that run isolated from the rest of its host, but utilize the same hardware and kernel resources as the host OS. This means that processes are fully isolated from the host's processes, and still they behave as virtual machines without the virtualization overhead in processing and computing capabilities. In addition, containers are usually light-weight processes and can be easily deployed, which makes them a flexible and easy to manage tool to be orchestrated.

Container orchestration is the process of automating the deployment, management, and provisioning of containers, from simple single-process containers to fully containerized distributed applications. This provides total control of containers during their life-cycle. When it comes to the Edge containerized architectures, they

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

are usually running very specialized applications with Machine Learning models, performing inference on the Edge and streaming data and predictions to larger systems. This makes orchestration a key process on Edge architectures, given the importance of having scalable and fault-tolerant systems which are not to be monitored by humans and expected to work as autonomously as possible.

Most container orchestrators are operated by defining all the configuration aspects of an application using YAML or JSON files. These configuration files define the **desired state** of the cluster, and the orchestrator will provide the resources specified in the config file to match the desired state with the actual state of the cluster. Then, containers are deployed and connected together through IPs or DNS services, and the applications are kept running by re-deploying failed containers, re-scheduling containers running on failed nodes, and scaling the application if the workload increases.

Many container orchestrating tools are already available, some being more widely adopted than others and differing in functionalities, but broadly speaking a container orchestration a 4-step process:

- Upload container images with the specifications of the applications and code to be run on the containers to an accessible image registry.
- Provisioning of containers with the storage, computational resources, infrastructure and files.
- Providing containers with a secure network to access other containers or external requests.
- Monitoring and management of the overall application. This step involves all of the scheduling, scaling, networking for service exposure, updates, and failure recovery.

During the FRACTAL project, and specifically during the course of T5.4, several container orchestrators have been studied, looking for an orchestrator which meets specific properties required in the Fractal platform:

1. **Lightweight:** Containers are usually running very lightweight applications in the form of microservices which can then be scaled to meet increasing resource requirements. Having a heavy orchestrator would introduce overhead in processing and networking which is not feasible to have on the Edge resource-constrained nodes.
2. **Deployable on the Edge:** IoT systems, and in particular the MPSoCs and FPGA boards being used in the Fractal platforms, usually have restrictions and differences with respect to other systems like Cloud computers and processors. These usually are linked to the processor architecture, being ARM64 the most typical architecture in the IoT domain, but also RISC-V being required to be supported.
3. **Automated and scalable:** Edge deployments are preferred to be as automated as possible to avoid human interaction with the deployed systems, reducing maintenance costs and increasing the functioning time of

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

the platforms. This is the reason why the orchestrator must be robust enough to self-heal in case a node goes down, provide high-availability, scale workloads when these increase in computational cost, and dynamically reschedule workloads from nodes that go down into available nodes.

4. **Easy to deploy**, but providing complex configurations: The ease of deployment is an important feature for an orchestrator, especially when dealing with Edge deployments where deployment on emerging or new nodes must be automated. Orchestrators with complex deployments (such as bare-metal Kubernetes) are avoided, going for simpler distributions specifically designed for Edge devices, like MicroK8S or K3s, while providing the same main functionalities.
5. **Open-source** and with an active community

Considering these requirements, three container orchestrators were implemented and tested, increasing in their complexity and available functionalities. First, Docker Swarm was implemented, providing a simple orchestrator with some core functionalities already supported by Docker Engine, the most widely used application for running containers in IT at the moment. Then, Kubernetes emerged while looking for a more complex and complete solution which covers all of the previously described requirements, but was found to have a complex deployment and big resource overhead for Edge deployments. Finally, more light-weight distributions of Kubernetes, K3S and MicroK8S, were found to be the appropriate tools for Edge containerized deployments, providing easy installation steps and the main functionalities of full Kubernetes deployments.

An overview of both orchestrators is provided below, highlighting the main features and differences between them.

5.4.1 Docker Swarm

Docker Swarm is an open-source container orchestrator which is natively supported by Docker, to orchestrate clusters of hosts running Docker Engine, a very popular container creation and management tool. It presents a series of advantages with respect to other orchestrators, mainly its ease of use and simplicity of deployment.

A Docker Swarm cluster is made up of manager nodes running Docker Engine, and worker nodes which are scheduled containers and workloads by the masters. To start using Docker Swarm mode from a node with an already installed Docker Engine, just run ***docker swarm init*** and the Docker Engine will create a Swarm node and provide a token and endpoint for the rest of the nodes to securely join the cluster.

The main features that make Docker Swarm a good orchestrator choice for Fractal deployments are:

- **Decentralized design:** Support multi-node swarms with master-worker architectures for decentralized applications.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

- **Service abstraction:** A set of identical containers are abstracted into a Service, which is in charge of load balancing the charge between all the containers behind, ensuring the availability of the deployed applications.
- **Scaling:** The number of containers (or tasks) behind a given Service can be scaled up and down to match the system and application requirements. However, the up-scaling and down-scaling of Services is not automated and must be done manually.
- **Overlay networking:** A virtual overlay network is created between the Swarm hosts so that containers deployed in different nodes can communicate with this overlay network instead of being forced to communicate via public or external networks, reducing latency and increasing availability.

The downside of Docker Swarm is that these services **must** be stateless, this means that all the containers behind a service must be equal, and the Service will not provide capabilities for stateful containers, meaning that if the containers behind a Service diverge in functionality after their deployment, the Service will no longer behave as expected when different containers are called.

More features and information about Docker Swarm and Docker Swarm mode can be found at <https://docs.docker.com/engine/swarm/>

From the Fractal platform perspective, these are the main advantages and disadvantages that Docker Swarm provide as a container orchestrator:

Docker Swarm pros:

- Service abstraction
- Cluster mode in master-worker architecture
- Ease of use
- Already integrated in Docker Engine
- Setting up a cluster is straight-forward

Docker Swarm cons:

- The only load balancing strategy is round-robin
- Does not support stateful containers
- Has no customization options
- Auto-scaling not supported

Conclusion:

While Docker Swarm is focused on its easy-to-use and easy-to-deploy features, it lacks some functionalities which could be required in more complex container applications, like specific load balancing strategies and support for stateful containers. It could be enough however for simple applications running on the Edge which don't require complex setups and can be easily deployed, where Docker Swarm can be used to deploy a container network and orchestrate containers easily.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

Docker Swarm has proven to be the entry gate to container orchestrators, being a great tool for beginners or not very complex container applications.

5.4.2 Kubernetes

Kubernetes (K8S) is an open-source container platform originally developed by Google and donated to Cloud Native Computing Foundation (CNCF) in 2015. It is widely used to manage and orchestrate containers and containerized applications, scheduling containers across a cluster, scaling and managing the cluster overall state and health.

The main components of a K8S cluster are the cluster itself, made up by a group of computing machines (either physical or virtual), a control plane, which is in charge of assigning tasks and controlling the overall state of the cluster, the kubelet, a service running on the nodes that deploys containers through the provided images, and Pods, which are the core concept of container instances in K8S clusters. A Pod is a group of one or more containers deployed into a node, sharing a pool of resources provided by K8S, an IP address, hostname, DNS resources, etc.

The advantages and disadvantages K8S offered to the Fractal platform are:

Advantages:

- Offers a wide range of functionalities, Pods are abstracted into Services, which then are exposed through Ingress and Load Balancer objects.
- Provides self-healing capabilities.
- Has lighter distributions for IoT and Edge deployments.
- It is very rich in features, plugins and allows integration with many other open-source tools.
- Automatically scales workloads.
- It is independent from Docker Engine, what gives complete freedom to choose a container runtime (Docker could be used, but it's not required).

Disadvantages:

- It is hard to learn, and managing clusters requires expert knowledge.
- Security is not enforced by default and requires external configuration.
- Its bare metal deployment is too heavyweight.
- It takes time to deploy even a simple application, and it usually ends up on heavy YAML syntax.
- Some of the plugins are required, like installing an external Pod Container Networking Interface (CNI).

Conclusion:

Deploying K8S requires a deep understanding of the underlying concepts and K8S objects, and their interactions with containers. Some of these concepts are analogous to Docker Swarm's (services, tasks), but ultimately K8S allows for a very complete configuration in networking and scheduling, and can be used to deploy

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

large containerized applications no matter how complex they are. Thus, application deployments can be customized to match the user needs.

Although deploying K8S and installing it on bare metal systems requires time and is a complex task, especially on Edge deployments, there exist light-weight solutions which are self-contained and can be installed easily, providing most of the functionalities of full K8S. These are MicroK8S and K3S, and both these tools have been deeply studied and implemented in WP6 (T6.1 and T6.2), and although they are out of the scope of this deliverable, they are worth mentioning because installing them mitigates one of the major downsides of K8S which is the installation process.

5.5 Self-orchestrating systems

As stated in Section 5.2 Container Orchestration, containerized applications are abstracted into code through container images, configuration YAML or JSON files, and then the overall application is given to the Container Orchestrator (for example, K8S), which will deploy the application and make sure that the desired state provided through the YAML matches the actual state of the application.

The next step in orchestration is to create systems which are able to orchestrate themselves, without needing any external user interaction after being deployed. These systems can be groups of containers that monitor the overall status of the system, resources, storage and availability, and take decisions based on the individual state of the host they are running on and the cluster status, attending to a set of rules that define what actions must be taken for each situation, defining alarms and controlling the cluster from the inside.

Take as an example a Pod which monitors the different nodes inside a K8S cluster. This “Monitoring Pod” is given certain permissions and is able to gather metrics and monitor the CPU, memory, and usage status of the whole set of nodes comprising the K8S cluster. This Pod runs a dedicated container which analyzes all the information gathered and makes decisions depending on the overall cluster status, by tainting nodes (applying restrictions like avoiding Pod scheduling on tainted nodes), rescheduling or scaling workloads, all from inside the cluster itself, with no need of a system administrator to monitor the state of the cluster.

Monitoring tools like Prometheus can also be used to monitor the overall status of the cluster in order to programmatically allocate resources or re-schedule workloads into the nodes, with a dedicated container running inside the cluster.

However, this is still an open problem and these self-orchestrating systems are to be optimized in the future. A custom orchestrator is being developed in WP6 T6.2 which will address this problem and provide a system that automates the monitoring and orchestration of the Fractal platform.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

6 MLBuffet v2

6.1 Introduction

There is a wide ecosystem of ML tools available that cover all the steps in the Machine Learning life-cycle. These ML tools usually cover these steps focusing on high-performance, low latency or optimized processing. It is clear how these tools are relevant in Cloud and high-resource environments, but there is still a lack of options when it comes to Edge deployments and IoT systems.

During the Fractal project, a necessity to have a system that is able to perform operations on ML models in Edge deployments and resource-constrained systems was spotted, while also having a good efficiency-to-cost computational performance.

During T5.4, a tool able to efficiently manage ML models was built under the project of MLBuffet. Several functionalities were progressively added to MLBuffet, from model inference on ONNX format models to model training and version control. The chosen deployment for the software stack was in the form of inter-communicating containerized microservices, so that new modules, potentially adding new or updating already existing functionalities can be included into the project.

6.2 Implementation

MLBuffet is an open-source, container-based Machine Learning Model Server for model inference, training and management. It has been specifically designed for Machine Learning projects that need to be carried out in the Edge, although it is also deployable on Cloud deployments. In this section the implementation details are described, showing the architecture and open-source tools used to build MLBuffet. All the code and installation steps are available at <https://github.com/zylklab/mlbuffet>.



Project	FRACTAL
Title	Specification of AI methods for FRACTAL system control
Del. Code	D5.5

As mentioned previously, MLBuffet is based on containerized microservices, each providing different functionalities and listed below:

- Inference: MLBuffet's Inferrer microservice can perform inference on ONNX and TensorFlow 2 developed models.
- Model management: Storage is a microservice which performs model storage, version control, and management. When a new model is uploaded to the server, it is stored in a dedicated container and managed, allowing downloading, re-versioning, modifying or deleting models.
- Training: Trainer module allows the users to schedule model training on Edge devices programmatically. This is done through a dedicated REST API resource where the user uploads the training script, dataset and required packages, and a new container is created performing the training operations, for any training Python library.

Code aspects and languages:

MLBuffet is entirely written in Python 3. This means that little code is necessary to implement fairly complex programs, compared to, for example, C++. This, together with its modular design, results in MLBuffet's code being lightweight (~160kB) and easily readable and expandable.

This tool consists of 6 modules: Inferrer, Modelhost, Trainer, Storage, Cache and Metrics. The core of each one is a Flask server, which expose several HTTP endpoints to communicate with each other.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

Deployment and environment:

MLBuffet was built to be entirely deployed inside a K8S cluster, as the application has full support for K8S. However, we open the door to integration with other orchestrators and programs, such as KubeFlow or orchestratorless Docker-Engine deployments. Thanks to K8S, the deployment of MLBuffet is easy and straightforward, and it can be escalated to simpler K8S distributions specifically designed for Edge devices, like MicroK8S or K3S.

In K8S each core module (modules which are not triggered or created by user-specific operations) is deployed through dedicated Services, which schedules scalable Pods with a Deployment, these are Inferrer, Metrics, Cache, and Storage.

For the side modules, Modelhost and Trainer, the Inferrer microservice is in charge of managing the creation and deletion of dedicated Deployments and Services for each model that is uploaded into the server. These models are individually packaged into a Modelhost Pod which exposes REST API endpoints to perform inference over the currently supported libraries. These makes the models totally decoupled from each other, ensuring full availability of each model, independently of the state of surrounding nodes or models.

The Trainer microservice is created whenever a user launches a training with its dedicated training script and dataset. This Trainer Pod will execute the training loop and will take care of its execution. When the model is trained, it is automatically uploaded to the model server for deployment, if able.


It must be noticed that these K8S object management is done through dedicated permissions granted on the Inferrer microservice, through a RoleBinding that allows it to manage Services, Deployments and Pods on the 'mlbuffet' namespace, ensuring the isolation of these permissions over external namespaces and avoiding security breaches.

MLBuffet also provides a Helm chart for automatic deploying of MLBuffet through this package manager for K8S. Its file structure is similar to K8S objects, with the only difference that Helm detaches constant values among all templates and gathers them into one file, for easy deployment configuration.

MLBuffet leverages distribution of workloads within a Kubernetes cluster to offer low response times, reliability and scalability. Furthermore, the deployment is hardware-agnostic, a key feature for IoT devices, which often combine different processor architectures in the same use case.

6.3 How does it work

Once MLBuffet is deployed (assuming a K8S deployment), the REST API of the Inferrer service is exposed. The K8S API can be queried about the Inferrer Service endpoints with the command `'kubectl get endpoints -n mlbuffet inferrer'`. Once the REST API is accesible to external users, it can be checked if it is available by

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

sending a simple curl request with `'curl -X GET http://<INFERRER_IP>:<PORT>/help'`. The help message should be displayed.

Updated information on the main functionalities of MLBuffet and how to perform operations over the server can be found at the [official GitHub](#), but an overview is given below:

Main features:

- Model Management:
 - o Upload a model
 - o Delete a model
 - o Update a model
 - o Get model information
 - o Download a model
- Model Inference:
 - o Get new predictions
 - o Cache duplicated predictions
 - o Perform inference asynchronously on ONNX or TF2 model
- Model Training:
 - o Train models on any Python ML Library
 - o Upload trained models
 - o Decoupled training from inference and storage
 - o Provide dataset, training scripts and required packages for training
 - o Train on Docker containers or K8S Pods

6.4 Installation

Installation of MLBuffet does not differ from a classical K8S application - building container images, uploading them and finally, deploying the application into a K8S cluster. All files and scripts for configuration and deployment of MLBuffet are located in the `"/deploy"` directory in the project.

MLBuffet provides a Shell script to ease image building through Docker's `"build"` command. Alternatively, one can build images manually, but taking into account that image names must match those in deployment `"YAML"` files, located inside `"autodeploy"` directory. Specifically, in `"kustomization.yaml"`, which defines constant values that will be used throughout the rest of deployment configuration files, such as microservices' image names. In case the images were uploaded to a remote repository, these names must point to the corresponding remote server's resource.

Once all images have been built and (if applicable) uploaded and desired configuration has been set, MLBuffet is deployed via `"kubectl apply"` command, indicating the directory of deployment files.

To test if MLBuffet is working correctly, there is a test endpoint available for microservices `"Inferer"`, `"Modelhost"` and `"Storage"`, through a GET request to `/api/test:8000`.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

6.5 Other implementations

6.5.1 ARM architecture (VERSAL node)

At the time of submitting this Deliverable, MLBuffet latest version is MLBuffet v2.0, which standardizes the software packages and base images being used to build the container images for ARM and x86 architectures. ARM is a very popular architecture for IoT and Edge devices, and it is crucial for MLBuffet to be able to be built in ARM systems, especially if it is targeted to be supporting the High-End node reference Fractal platform (VERSAL board).

MLBuffet v2 can be either built for x86 and ARM architectures using the Docker Engine image build functionality, and images can be deployed on both processing architectures without issues.

For RISC-V architectures, every software package being used, from ML libraries to APIs and databases should be built from source for RISC-V architectures, and most the open-source tools used by the microservices are not yet available for RISC-V.

For this reason, MLBuffet is not yet supported for RISC-V architectures, but as the RISC-V architecture becomes more popular as a standard open-source processor architecture and more open-source tools are available and built for these systems, it will be possible and is expected to build MLBuffet also for RISC-V machines.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

7 Conclusions

In this document, the main developments, tools and technologies that have been utilized for the development of the Fractal System Controller are presented and described. This document is a descriptive collection of methods to be followed and tools to deploy in order to operate ML pipelines and workflows on Edge devices.

Firstly, an overview of the MLOps methodologies is done, explaining how to effectively implement MLOps practices for a more agile development in ML tool chains. These methodologies allow the Use Cases to continuously deploy and integrate their developments into the Fractal Edge node, thus minimizing the down times and performing system and software updates in a more agile and resilient way.

Then, the orchestration concept is introduced and explained from the theoretical perspective, giving the insights on how orchestration is required for the automation and autonomy of the Fractal Edge Node. Orchestration strategies described are the bottom-up and top-down alternatives, which address both the decentralized and centralized computing paradigms respectively. Once the insights of theoretical orchestration are given, container virtualization is introduced as a reliable technology to package and deploy software in artifacts which can be effectively orchestrated by Container Orchestrators. A comparison and analysis of the two main container orchestrators that have been researched during the task is given: Docker Swarm and Kubernetes are two options to be used by the Use Cases which choose to manage their deployments in a containerized way, and the pros and cons of each one are analyzed to help the users decide between one or the other.

Finally, MLBuffet is presented and described as a containerized tool which can be deployed on one or more Fractal Edge nodes to cover all the main steps addressed on the Edge in the ML models life-cycle, from model storage and version control, Edge training and Edge deployment. Its containerized and microservice-based architecture makes it an intrinsically orchestrable tool, and it can be utilized as the core service in the Fractal system control to manage all the ML processes to be performed on the Edge. An overview of its functionalities, architecture, and installation steps are provided, although more details and the latest versions can be found on the provided public GitHub repository.

During T5.4, the efforts were focused on translating the research done during T5.1 into practical implementations, specifically on the concepts of Orchestration and Adaptability. As a result, the MLOps methodologies detailed in this deliverable together with the release of MLBuffet allow the Fractal Edge Nodes to have a set of tools that dynamically perform all the steps in the ML workflow. Application containerization and the usage of Container Orchestrators (K8S or Docker Swarm) also were introduced into the project's available technologies, providing a proactive adaptation environment for the tasks to be done in WP6 and providing the UCs with a complete set of technologies.

	Project	FRACTAL
	Title	Specification of AI methods for FRACTAL system control
	Del. Code	D5.5

8 List of figures

Figure 1: Valid photo examples.....	10
Figure 2: Example of labeling different types of defects.....	10
Figure 3: Example of labeling considering only cracks and fissures.....	11
Figure 4: Example of a MLOps pipeline.....	12
Figure 5: MLBuffer architectural implementation.....	22



Project	FRACTAL
Title	Specification of AI methods for FRACTAL system control
Del. Code	D5.5

9 List of tables

Table 1 - FRACTAL Project objectives.....	5
Table 2 - List of available data labeling tools.....	9



Project	FRACTAL
Title	Specification of AI methods for FRACTAL system control
Del. Code	D5.5

10 List of Abbreviations

AI	- Artificial Intelligence
API	- Application Programmable Interface
CI/CD	- Continuous Integration & Development
CNCF	- Cloud Native Computing Foundation
CNI	- Container Networking Interface
CSV	- Comma separated values
DevOps	- Development Operations
FPGA	- Field Programmable Gate Array
IT	- Information Technologies
JSON	- JavaScript Object Notation
K8S	- Kubernetes
ML	- Machine Learning
MLOps	- Machine Learning Operations
MPSoC	- Multi-Processor System on Chip
NLP	- Natural Language Processing
REST	- Representational State Transfer
UC	- Use Case
WP	- Work Package