

Data Analytics on Snap4City, Machine Learning Operation MLOps on Snap4City via ClearML CPU/GPU - HPC

From Snap4City:

- Development Life Cycle user manual:
 - <https://www.snap4city.org/download/video/Snap4Tech-Development-Life-Cycle.pdf>
- See Client-Side Business Logic Widget Manual:
 - <https://www.snap4city.org/download/video/ClientSideBusinessLogic-WidgetManual.pdf>
- Videos and PDF of Training slides <https://www.snap4city.org/944>
- You may read the TECHNICAL OVERVIEW,
<https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>
- <https://www.snap4city.org>
- <https://www.snap4solutions.org>
- <https://www.snap4industry.org>
- <https://twitter.com/snap4city>
- <https://www.facebook.com/snap4city>

Coordinator: Paolo Nesi, Paolo.nesi@unifi.it

DISIT Lab, <https://www.disit.org>

DINFO dept of University of Florence,

Via S. Marta 3, 50139, Firenze, Italy

Phone: +39-335-5668674

Access Level: public

Date: 10-09-2025

Version: 1.3

Table of Contents:

1.	Overview	4
1.1	– Data Analytic Processes Possibilities	6
1.2	– Case B: Snap4City with DAP Container Manager, No MLOps Support	8
1.3	– Case C: Snap4City with MLOps Support, & DAP Container Mng: ML/AI.....	11
1.4	– BEFDIT (Behavior Execution Framework for Digital Twins).....	14
1.5	– Creation of Smart Applications using ML/AI supports.	17
2.	MLOps in ClearML	22
2.1	– Managing an Experiment	22
2.1.1	– Credentials	22
2.1.2	– Initialization of an Experiment	22
2.1.3	– ClearML import.....	22
2.1.4	– Task Creation/Init	22
2.2	- Python Version and Package compatibilities in ClearML Agents.....	23
2.2.1	– Problems of compatibility among versions e packets Python.....	23
2.2.2	– Selection of Docker Image of the Agent for CPU Compatibility	24
2.3	- ClearML Basic Functionalities	24
2.3.1	- Log of the Hyper-parameters	24
2.3.2	- Log Artefacts	24
2.3.3	- Add Artefacts	24
2.3.4	– Use the Artefacts	25
2.3.5	- Models	25
2.3.6	– Load of a model.....	25
2.3.7	- Log Metrics.....	25
2.3.8	– Data types registered with Logger	25
2.3.9	– Automatic Recording of metrics	26
2.3.10	- Manual Recording of Metrics.....	26
2.3.11	- Pipelines	27
2.4	- ClearML Data	28
2.4.1	- Dataset Versioning.....	28
2.4.2	- Dataset Organization and Access	28
2.4.3	- Dataset Documentation and Tracking	28
2.4.4	- Using ClearML Data	29
2.4.5	- ClearML Data CLI	29
2.4.6	- ClearML Data SDK.....	31
2.4.7	- Graphic User Interface	33
3.	ClearML - Experiment Comparison	35
3.1	- Experiment Table	35
3.2	- Experiments Leaderboard	35
3.3	- Select Multiple Experiments.....	36
3.4	- Detailed Comparison	37
3.4.1	- Comparison Modes	37
3.5	– Examples of Python for testing JupiterHUB -> ClearML execution	42
3.5.1	– example 1.....	42

3.5.2 – example 2.....	43
4. Hyperparameter Optimization.....	44
4.1 - Supported Optimizers.....	46
4.1.1 - Comparison task	46
5. ClearML Feature Testing.....	49
5.1 - Dataset Versioning.....	49
5.2 - Comparing Experiments.....	49
6. Inference API	51
6.1 On-Demand API.....	51
6.2 API Task Enqueue/Sporadic	52
6.3 ClearML Utils	52
6.3.1 - Technology Used: Stack TALL and Filament.....	52
6.3.2 - Advantages of Filament.....	53
6.4 Main ClearML Dashboard	53
6.5 ClearML Serving Machine Management	53
6.6 On-Demand Endpoint Management	54
6.7 Task Management for the Task Enqueue/Sporadic Service.....	55
6.8 Service Call Logs	56
7. Use of Services AI/DA from IoT App/Proc.Logic	59
7.1 Authentication with Snap4City	61
7.2 Remote task notification system.....	61
7.2.1 - Executing the Task in Service Mode	62
7.2.2 - Integration with Skype	62
7.2.3 - Starting the Task.....	62
7.2.4 - Advantages of the SkPy Library:	63
8. Testing and Validation	64
8.1 Model Development and Training	64
8.2 Deploying the Model on ClearML Serving	64
8.3 Testing with Node-RED Block	64
8.4 Testing On-Demand Inference API with CSBL	65
8.5 Model Monitoring	67
9. Example Scripts	68

1. Overview

The design and development of **Data Analytics, DA, Processes (DAP)** is mainly performed taking in mind that their development cases are performed in Python or Rstudio. For DAP we intend the development of algorithms for some computation, data processing: KPI, predictions, optimization, simulation, etc., exploiting ML (machine learning), AI (artificial intelligence), XAI (explainable AI), operating research, statistics, heuristics, LLM, NLP, etc.

The DAPs can be devoted performing tasks of model training, model execution, computation, simulation, etc., in batch or stream, on demand or as a stable process serving requests on demand from some API. The design of DAP implies to decide their aims, for example, for implementing specific algorithms, or making predictions, anomaly detection, suggestions, statical analysis, clustering, recognition, counting, classification, object detection, KPI estimation, optimization, conversion, etc. Most of these aims can implemented by using techniques as ML, AI, XAI, NLP, LLM, operating research, statistics, etc. To this end they would need to exploit a set of libraries for Python or RStudio to produce a model (in a training phase) which in turn has to be saved to be later exploited in execution/inference. Python and RStudio platforms may exploit any kind of libraries such as Keras, Pandas, and hardware accelerator as NVIDIA to use Tensor Flow, and clusters of CPUs/GPUs, HPC infrastructures, via ClearML, MLOps, etc., exploiting Kubernetes or other solutions.

Moreover, in order to get data, the DAP in Snap4City can access to any kind of storage from external services and can access to the Snap4City KB (knowledge base, service map) and Big Data store. In that case, the access to Snap4City data is GDPR compliant and thus respect the privacy, the data licensing by using authenticated Smart City APIs, via some Access Token as explained in the **Development Life Cycle** Manual referred in the cover. The platform allows the access to historical and real-time data, and permits to save the resulting data produced by the algorithms, for example, heatmap-related predictions, the assessment of data quality, traffic flow data, ODMs, labels of detected anomalies, LLM requests and results, etc., also using some specific APIs.

For the analysis details are reported in the **Development Life Cycle** and for a **DAP** one should identify:

- What process must be implemented by the DAP?
- Which data models would be consumed and produced?
- Which data are needed, to be consumed?
- The DAP to be implemented is for training or for production?
- How many users are going to exploit the DAP at the same time? How many executions per minute or per day, the usage is sporadic or periodic?
- How many processes for production I am going to have at the same time?
- How big would be the AI model to be loaded for execution the first time?
- From where the DAP is expected to be called, from a Dashboard/view? or simply from a back-office process as a MicroService?
- Which is the expected execution time?
- Which is the expected precision, and which is the state of the art?
- Do I need to execute the DAP exploiting special hardware as NVIDIA since I am going to use CUDA, tensor flow, ...?
- How much GPU I need?
- How much memory to load the AI model I need?

How to proceed to design the single DAPs according to its nature?

Here in the following the most relevant tasks summarized, just to recall you the main aspects to be addressed:

- Problem analysis, business requirements.
- Data Discovery, Data ingestion, data acquisition (as above presented that can give for granted),

data access from Snap4City platform or from other sources.

- Data set preparation, transformation, identification of features, normalization, scaling, imputation, feature engineering, etc., eventual data ingestion to the Snap4City platform by using Proc.Logic or Python and then storing data in the storage. The process of feature engineering may be performed by also using a PCA (principal component analysis or other techniques), or directly performing the first training and assessing the relevance of the features (for example with some RF, or using some XAI or other tech.), may be discharging those less relevant.
- Target Assessment Model Definition (mandatory to assess the precision of the results, the quality of the solution produced)
 - Identification of metrics for the assessment of model results, KPI.
 - Typically: R2, MAE, MAPE, RMSE, MSE, MASE, MAPE, ...
- Screening on Models/Techniques, for each Model/Technique or for the selection Model/Technique perform the
 - Model/Technique Development/testing
 - Performing for each of them some hyper-parametrization, take care about the range of parameters with respect to the obtained best value of each of them;
 - Defining a good loss KPI according to the target results to be achieved and controlling the loss trend.
- Best Model/solution selection among those tested
 - If needed reiterate for different parameters, features, etc.
 - Comparison with state-of-the-art results.
 - Needs of Explainable AI solutions: global and local.
- Deploy best Model/solution in production, monitoring in production. In this phase assumes particular relevant:
 - Assessing performance and resources needed: GPUs, Memory, storage, etc.
 - Security of data and solution
 - Scalability of the solution, in terms of multiple users requesting the same computation,
 - multiple requests of the same computation but working on different spatial area, such difference cities, KB, maps, graphs, time series, etc.

In conclusion, the main activities are those of **Development** and **Execution**.

1.1 – Data Analytic Processes Possibilities

According to the kind of **DAP support provided** by the Snap4City platform you are using, the develop and the execution of DAP solutions can be performed and enforced in different manners, but it is any way possible to put in execution your DAP on Snap4City. The Snap4City DAP support is provided by means of a few different solutions which can be classified according to the components installed, which may impact the activities of **Development** and **Execution** in different manners.

The main components are:

- **Local Development environment** on your premise for Python and/or Rstudio.
- **Jupyter HUB Server:** a server providing development environment for Python with web interface, totally integrated with Snap4City platform as SSO.
- **R-Studio Server:** a server providing development environment for Rstudio with web interface. Which may be totally integrated with Snap4City platform as SSO
- **DAP Container Manager:** A solution for statically or dynamically creating Containers including DAPs and putting them in execution on cloud. It can be based on Marathon/Mesos as well as Kubernetes or others. **This is solution to be used for allocating simulation tool, for example.**
- **MLOps Support:** A solution and tools to support developers in creating their DAPs, making experiments, optimising, testing and validating them, keep track of the performed experiments, etc., and also putting them in execution on some Container, exploiting also clusters of CPU/GPU, HPC, and dynamically creating containers and processes on the infrastructure. They can be stable services or activated on demand.
- **IoT App/Proc.Logic processes:** A Node-RED + Snap4City Libraries process which can be installed on edge, on premises or on cloud, which can exploit the Snap4City facilities: authentication and authorisation, data ingestion, data transformation, management of DAPs, calling of DAPs API, also calling service on MLOps via their APIs, interacting with dashboards (server-side business logic), interoperating with any kind of protocols and formats.
- **A&A,** Authentication and Authorization mechanism of Snap4City or that of others interoperable platforms, SSO, in OAuth.
- **Advanced Smart City API, ASCAPI:** a set of APIs to access/provide data from/to the Snap4City platform, as REST Call, microservices.
- **API Manager of Snap4City,** a solution for accessing to API (produced by DAP Containers static or dynamic, as well as from MLOps services stable and/or on demand). The API Manager also may enforce rules for the consumption of API call, for example providing the authorisation to a given user to consume 100 API calls to a specific service per day and controlling its enforcement and consumption.

The main cases can be (starting from the less comprehensive to the most):

A. Snap4City platform having: No DAP Container Manager, No Jupyter HUB Server, No R-Studio Server, No MLOps Support. In this case, the developers can develop their **DAPs** in the language they prefer, on some server or on their laptop.

1. Once developed the DAP, it can be exploited by the Snap4City platform by making the DAP accessible via some API (as static process), or by using some data exchange via database or other means which can be controlled and exploited by some IoT App/Proc.Logic or Dashboard/View (mapping the API on some FW). If the DAP exposes some APIs, we suggest using Flask for Python and Plumber for Rstudio. In this case, the IoT App/Proc.Logic/Dash has the possibility to call the DAP as an external service.
 - i. The external DAP providing the API may be protected by some external A&A mechanism, you may need to pass it the Access Token. The IoT App/Proc.Logic can be connected using them.
 - ii. Please note that in the case of using External APIs from dashboards/views in JavaScript from client side, you may need to expose the credentials on the web page,

avoid passing some Access Token. In alternative, we suggest calling the external services APIs only from the IoT App/Proc.Logic.

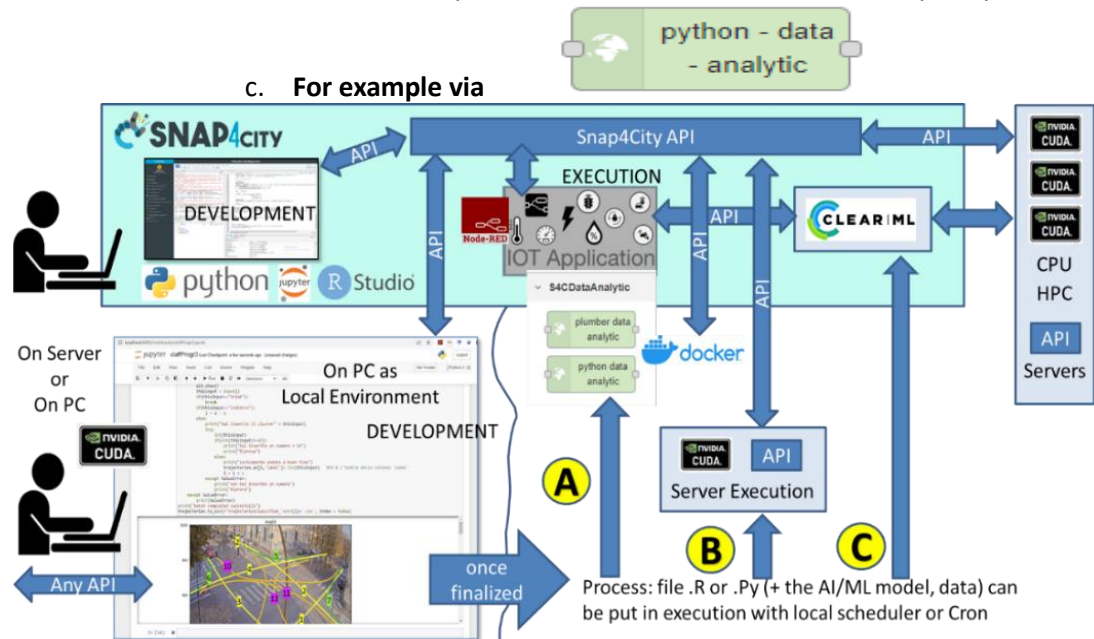
2. DAPs can exploit the Advanced Smart City API, ASCAPI, of Snap4City according to the Development Life Cycle. DAPs can access to protected data according to A&A based on OAuth as Access Tokens and GDPR, and can send data for their ingestion and save them into the platform, etc. The usage of the APIs is described in the Development Manual.
 - i. The A&A for data access/save from IoT App/Proc.Logic is automated by the Snap4City Libraries and can be performed one on Edge, and totally transparent for the IoT App/Proc.Logic on cloud of Snap4City platforms, from MicroX to large solutions.
 - ii. The A&A for Snap4City Dashboards/views is also automated and may have JavaScript developed as **Client-Side Business logic**, see reference manual mentioned on cover of this document.
 - iii. The A&A for third party applications can be developed according to the Development Life Cycle manual.

B. Snap4City platform with DAP Container Manager, No MLOps Support. In this case, the **DAP Container Manager** is integrated into the Snap4City platform accessible (typically based on Marathon/Mesos, or Kubernetes). For the final users and for the developers the usage of one kind of DAP Container Manager or another it is not relevant and has no impact.

1. **DAP Container Manager based on Marathon/Mesos (progressively deprecated):** provided on Snap4City.org. The developers have to code DAPs as API based processes, which expose their APIs via Flask for Python and Plumber for Rstudio according to Snap4City directives, see Development Life Cycle Manual and examples on web portal and training course.
 - i. The developers may have access to one or more **Jupyter HUB Servers** and **R-Studio Servers**, for DEP developing, or can develop the DAPs on their laptops/desktop. This means that the activities of tuning, hyper-parametrization, validation, etc., are all performed by coding.
 - ii. **Once a DAP is developed** according to the Snap4City directives for DAP development, it could be put in Execution. To this end, the:
 1. **(B) in the following figure: DAP is put in execution on some server to expose the APIs** which can be used by any IoT App/Proc.Logic as in **A.1 case**, above. In this case, the DAP can exploit the direct resources of the server, even NVIDIA boards, HPC, etc., if provided. In the cases of Snap4City.org, these kinds of DAPs can exploit (i) a large number of NVIDIA servers with huge number of GPUs, (ii) use external API of third party, (iii) exploit the Smart City APIs.
 - a. The API based DAPs could be made accessible for Dashboards exposing the API on Internet. On the other hand, this may create a door for eventual attacks and unauthorized access to the DAPs.
 - b. The API based DAPs should be protected in some manner. For example, working only if the DAP receives a valid Access Token (taken from the section), by which it can access via ASCAPI to protected content.
 2. **(A) in the following figure: DAP code is loaded on the DAP Container Manager** Marathon/Mesos via **special Snap4City DA nodes for IoT App/Proc.Logic** (available on IoT App advanced, for developers, typically accessible for AreaManager users). Those nodes request to the DAP Container Manager to automatically create a container and allocate it statically on cloud. Please note that, each new DAP has a counterpart node-red node into the **IoT App/Proc.Logic flow which created it** and is realized as a new container. The container of the DAP is only accessible/visible for the user who created it (which can list them on the IoT App/Proc.Logic list,

where it can also be deleted/managed).

- a. This approach is suggested to be used only for realising prototypes and not for realising stable production DAPs due to its limited scalability and high consumption of resources. Moreover, the DAP container in this case is usable only by the IoT App of the user who create it.
- b. DAP may use the NVIDIA support only if provided at cloud level on any DAP container. Images of DAP containers need to be customized for adding specific libraries, and the exploitation of NVIDIA boards. Please note that this approach on Snap4City.org does not allow to the DAP to exploit the NVIDIA cluster facilities of Snap4City.



2. DAP Container Manager based on Kubernetes, in this case we have two cases:

- i. Static allocation of containers including your DAPs and exposing some API, putting them on Kubernetes. They can be controlled by API Manager.
- ii. Dynamic allocation of Containers including some DAP/Simulation (for example SUMO). They can be controlled by API Manager. The deploy of container is performed by Snap4City Container Orchestrator that can work directly on Kubernetes and has presently some limitations on container kinds.

C. (C) in the above figure: Snap4City platform with MLOps Support and its integrated DAP Container Manager. This case represents the most advanced solution for dynamic DAP **development** and **execution** as described in detail in this document and described as concept in **Section 1.3** and **Section 1.4** for the BEFDIT (Behavior Execution Framework for Digital Twins).

1.2 – Case B: Snap4City with DAP Container Manager, No MLOps Support

In this **case B**) depicted in the above figure and described above, the data scientists may develop their DAPs on the provided Jupyter HUB, as a Python development environment, as well as on Rstudio Servers. In this kind of Snap4City platform, the development of DAPs can be performed on Jupyter HUB in Python as well as on Rstudio Servers by exploiting:

- APIs provided by Snap4City, as ASCAPI or provided by some DAP deployed in some manner, in this case the Jupyter HUB can be on a CPU or GPU server of Snap4City platform you have. You can even executed on your PC and using the credential to access to the Snap4City public APIs only;
- APIs of third party, not accessing to the resources CPU/GPU of the Snap4City platform you have.

In Snap4City, the access to Jupiter HUBs for Python and/or Rstudio Servers for the development of DAP is provided by the RootAdmin. The role of the Snap4City users has to be AreaManager or higher.

Please note that, in this case, the activities of training, optimisation, hyper-parametrization, experiment tracking, assessment and validation, comparison, tuning, etc., are all in the hands of the developers which cannot use the provided MLOps facilities of Snap4City.

On the other hand, the activity to put DAP in production is simplified. In the sense that, the DAP can be taken in charge by the DAP Container Manager for the execution.

The DAPs can be executed on:

- **Dockers Containers** accessing and controlling them via some API, and these can be automatically produced and manage by the platform.
 - **In this case**, the management is typically performed by some Proc.Logic (IoT App). Please do not produce and make accessible API from Node-RED, is possible but unsecure and not scalable.
 - The containers are automatically allocated on cluster and maintained alive to be used by Marathon (with some limitation on Kubernetes) and may exploit the GPU/CPU according to the HD accessible and configurations. They are usually allocated dynamically, and they are moved from one VM to another by the DAP Container Manager.
- **Dedicated servers for developers** and leaving them to access to the storage for using the data and providing results via Snap4City API, in authenticated and authorized manner.

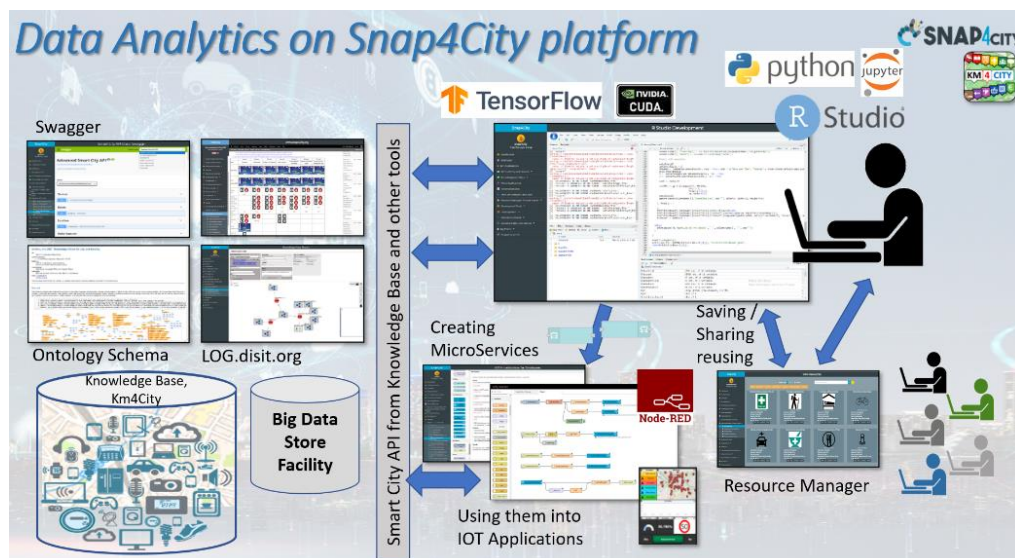


Figure – Schema of DAP/Data Analytics (ML, AI) development to be used as permanent Containers (exploiting CPU/vGPU on cloud)

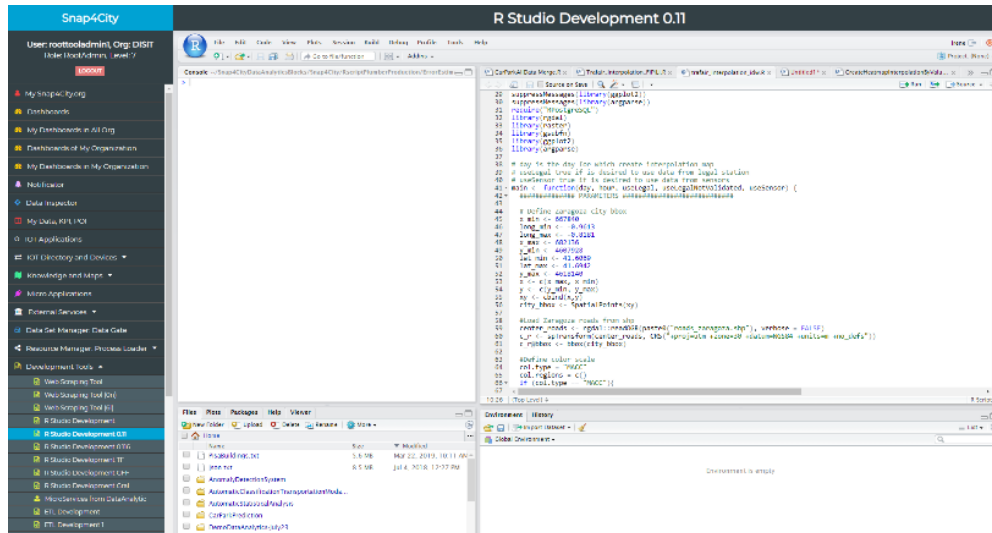


Figure – DAP development in R-Studio, similar to Python which is in Jupiter HUB.

In Python and/or RStudio cases, the script code has to include a library for creating a REST Call, namely: Plumber for RStudio and Flask for Python. In this manner, each process presents a specific API, which is accessible from an IoT App/Proc.Logic as a MicroService, that is, a node of the above-mentioned Node-RED visual programming tool for data flow (API can be mapped and managed by the **API Manager of Snap4City**). Data scientists can develop and debug/test the DAPs on the Snap4City cloud environment since it is the best way to access at the Smart City API with the needed permissions. The source code can be shared among developers with the tool “Resource Manager”, which also allows the developers to perform queries and retrieve source code made available by other developers.

Rstudio and Python DAPs may include conceptually any kind of libraries for ML, AI, operative research, simulation, etc, and may load AI models, LLM, etc. On the other hand, when the process is adopted to produce a container, as in the next figure, the container has to include the library used in the code. The Development environment may be configured to allow at the single operators to load their own preferred libraries. Or requested libraries in the containers may be added by the RootAdmin. This can be performed by requesting a specific image to the platform manager and indicating the library you would like to have on Container executions.

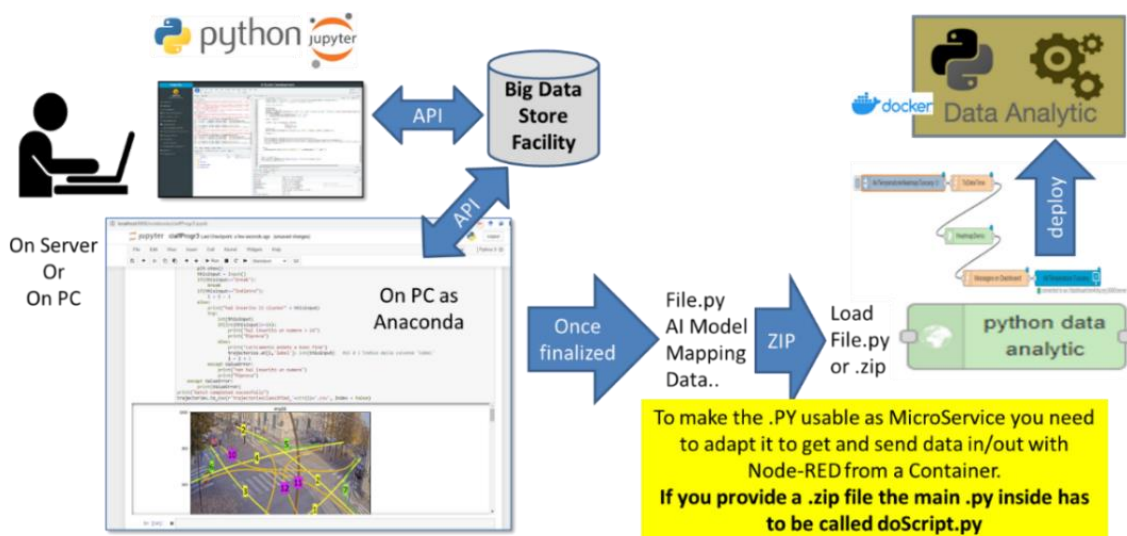
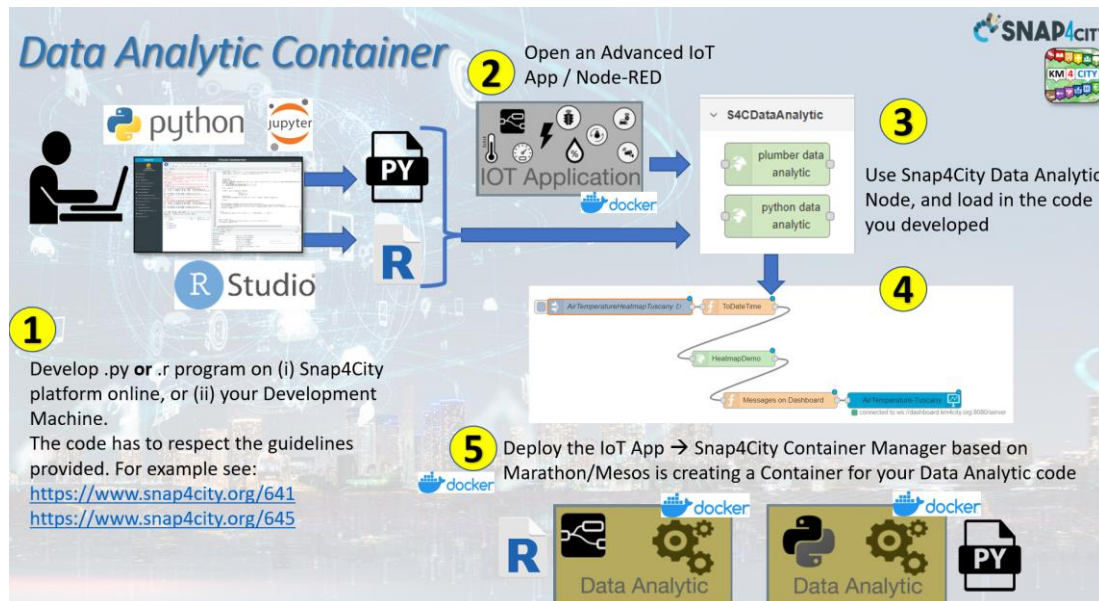


Figure – Case B) DAP development flow in Python, from a Jupyter hub as well as from PC with Anaconda development environment installed. Please note that you can load you AI model putting it into the ZIP loaded on the block to be used for the deploy of the container on the Marathon cluster.

This description of the flow refers to case in which the Python or Rstudio are created to be used as

MicroServices from a Proc.Logic/IoT App. An alternative is to develop the DAPs to be used as standalone services, working on API, or providing some REST Call, and thus usable from Proc.Logic/IoT App according to the API or by collecting results on database. These aspects are described into the training course.



Figure– Data Analytics development flow in Python and integration into Proc.Logic / IoT App.

In Snap4City, there is a specific tutorial for the Data Analytic development with several examples:

<https://www.snap4city.org/download/video/course/p4/>

Read the mentioned slide course and/or platform overview to get a list of Data Analytics in place:

<https://www.snap4city.org/download/video/Snap4City-PlatformOverview.pdf>

We also suggest reading the Snap4City booklet on Data Analytic solutions.

https://www.snap4city.org/download/video/DPL_SNAP4SOLU.pdf

Read more on: <https://www.snap4city.org/download/video/course/p4/>

If you are interested to develop ML/AI processes with or without MLOps support, there is Python library which can be obtained only via subscription please contact snap4city@disit.org

1.3 – Case C: Snap4City with MLOps Support, & DAP Container Mng: ML/AI..

In this case C) of the above list, the solution provided by Snap4City includes the support for MLOps, Machine Learning Operation. In Snap4City, the MLOps is provided by using a custom version of ClearML tool, using a Jupiter HUBs for Python to develop DAPs, using Kubernetes or on former Marathon and Mesos or exploiting direct clusters of CPU/GPU and HPC, or combination of the Kubernetes and the latter. The access to this facility to the DAP developers is provided in any Snap4City platform by the RootAdmin to AreaManager role of users or higher. Snap4City.org is the most complete platform with clusters of CPU/GPU and HPC with both Marathon and Kubernetes; and Mobility4Future.com totally based on Snap4City has the configuration with Kubernetes and MLOps for accessing to HPC of Lutech.

Snap4City with MLOps facility fully supports the phased of **Development** and **Execution** as described in the following.

Development, with the activities of:

- Training with different parameters and models to be trained, hyper-parametrization, tuning, etc.

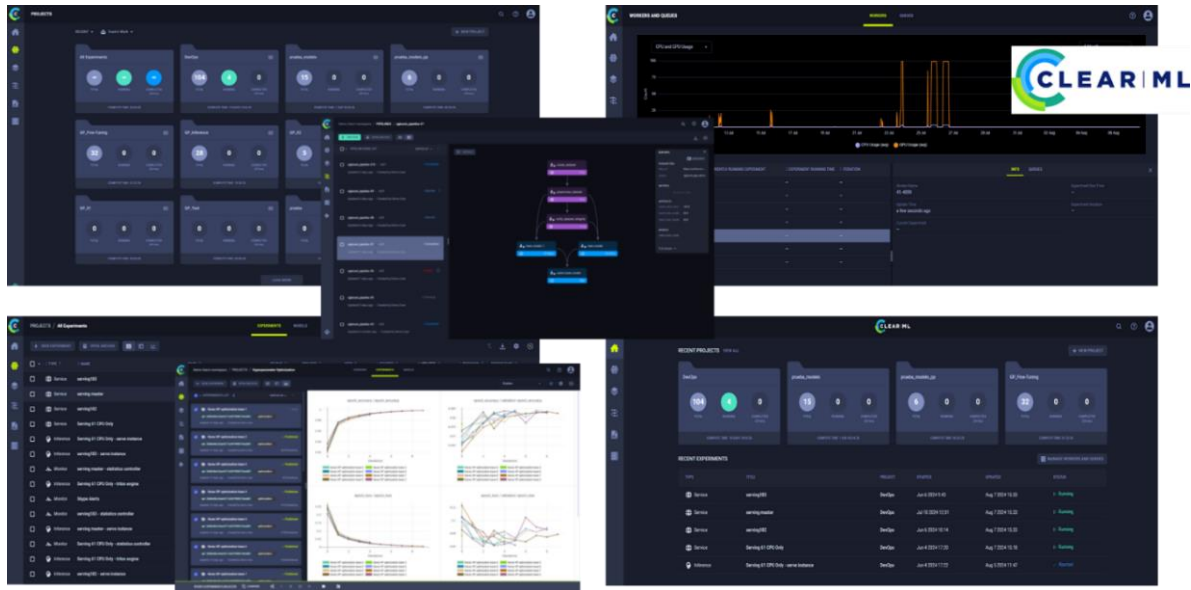
- Validation and test in batch to find the best results wrt metrics, tracking and comparing the experiments, etc.
- Managing high computational costs, managing time consumptions, sending DAP automatically on free GPU/CPU of clusters and/or HPC, etc.
- And many other functionalities as described in the following
- On this phase, Snap4City.org provides access to a Jupyter HUB from which it is possible to develop the Python coded DAP, exploiting ASCAPI or third-party APIs, and send them on MLOps Support, performed in ClearML, to exploit a number of clusters in CPU/GPU with many kinds of NVIDIA boards: H100 NVL, GV 100, RTX 4090, RTX 3090, Titan XP, etc.
- Mapping API produced on API Manager of Snap4city to make them accessible from Dashboards and web/mobile apps and controlling their consumption. (Optionally)

MLOps is realized by using ClearML, which has as main features:

- **Experiment Tracking:** Provides advanced features for experiment tracking, including automatic logging of metrics, output, source code, and the execution environment. This ensures that each experiment is reproducible, and its results are easily shareable and comparable.
- **Data and Model Management:** Provides tools for efficient management of datasets and models, allowing for easy versioning, archiving, and sharing. Users can track model versions and easily associate them with corresponding experiments.
- **Integration and Compatibility:** ClearML is designed to integrate with existing development environments and tools, such as **Jupyter Notebooks**, **TensorFlow**, **PyTorch**, and many others, thus supporting a wide variety of workflows and technology stacks.
- **User Interface and MLOps Dashboard** offers an intuitive dashboard that allows users to monitor the status of their experiments in real time, view metrics and outputs, and manage resources and execution queues, all from a single interface. Root user of ClearML has the possibility of observing the activities of all the users/developers.
- **Automation and Orchestration:** It allows the remote execution of experiments on any machine and distributes the tasks to be executed according to a system of queues and priorities. Also automating Hyper-parametrization via **Optuna**.

Please note that, the development is performed on Jupyter Hub in the personal space of the developer by enforcing a specific connection with the ClearML server (with the specific account of the Developer in the ClearML environment) by using specific credentials and code calls for data, and processes as described in the following. In the Snap4City.org version, for security reasons, only specific Jupiter Hubs can exploit the connection with ClearML, they are typically under progressive backup on cloud, while versioning is provided with SVN support. In principle any Python development environment could exploit such as connection, while open to all would not be safe enough.

These aspects are described in the rest of this document.



Execution on production (for ML/AI also called Inference phase)

The **Execution on production** has to guarantee support for:

- Security of data and DAP solution access, permitted only to A&A users. Also in this case, the developer, working on Jupyter Hub, can send the code to the MLOps only by using its specific credentials and IDs.
- Scalability of the DAP solution, in terms of multiple users requesting the same computation at the same time,
- multiple requests of the same time working on different spatial area, such difference cities, KB, maps, graphs, time series, etc.
- monitoring the resource consumption in the terms of memory, storage, and CPU/GPU clocks/percentage. Eventual early warning and alarms sent to administrator. Possible the accounting of resource consumed.
- Eventual block and removal of strange / non desired processes.

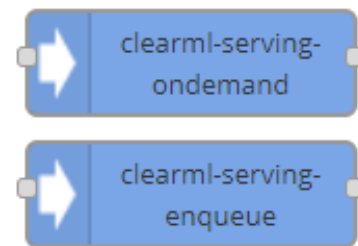
The Execution on production is enabled **by creating DAPs (with a modality described in the rest of this document) which can be called via some APIs (provided, made accessible) according to TWO Modalities:**

- **SPORADIC services for REST calls, which are allocated and executed every time they are called (former name Enqueue):** to call the API of a DAP which is created as a task and executed at every API call by the MLOps according to the list of requests put on some queue. The DAP is allocated automatically on some server as temporary container and process (NVIDIA / GPU, clusters or classic CPU clusters, or HPC, or Kubernetes) by the MLOps manager just for the single execution.
 - This means that each DAP Execution includes the loading time on boards/memory, and that the DAP does not remain in memory, and the memory of the servers (CPU/GPU) are not permanently booked for that DAP, the Kubernetes container is deployed, executed and destroyed.
 - This approach is suitable for DAPs which are executed sporadically, and/or periodically for which the overhead time to put them in execution is acceptable with the respect to the time for computing and delivery of the response, and the period of execution.
- **STABLE services static allocated REST Calls performing the on memory and GPU/CPU board allocation once, at the first Rest call (former name: OnDemand):** to call the API of a DAP which is created as a task into a container and load statically on the server (NVIDIA / GPU, clusters or classic CPU clusters, Kubernetes, HPC).
 - This means that at the first execution the time to load the process on memory of boards will be evident and may be relevant, suggest solution for LLM and large AI Models.

- This means that, once loaded, the DAP is ready to respond to the API call since is statically (permanently) allocated on the execution server, occupying memory (of CPU mem, GPU video mem) and not the actual CPU/GPU, until is not called (wake up) via API.
- This modality is particularly suitable to exploit DAPs which need a relevant time to be loaded and put in execution, thus making the usage of the **Sporadic modality** not viable. For example, the usage of a LLM needing 24Gbyte or more would need lot of loading time, with respect to its single execution time by using the OnDemand/Stable modality only in a few seconds. So that in this case, the Sporadic solution is not suggested.
- **On both these modalities**, Snap4City.org provides access to exploit a number of clusters of services and single servers in CPU/GPU with many kinds of NVIDIA boards: H100 NVL, GV 100, RTX 4090, RTX 3090, Titan XP, etc., and Kubernetes HPC based.

The Snap4City platforms with MLOps support, may expose APIs of the DAP in the two modalities of Enqueue/Sporadic and OnDemand/Stable which can be called in authenticated manner via API as well as via IoT App/Proc.Logic nodes, as reported on the right side.

The two nodes are accessible as a separate Node-RED library of Snap4City microservices: <https://flows.nodered.org/node/node-red-contrib-snap4city-clearml> which can be installed on any IoT App/Proc.Logic on cloud and on Edge.



Please note that, the DAPs accessible via **Enqueue/Sporadic or OnDemand/Stable** modalities can be called from external services as well if the API are mapped on **API Manager**, or statically mapped on some FW (not suggested). For security reasons they can be called only by using the current Access Token of the section for the user. This allows to access at the DAPs from CSBL and any Web Application which is developed according to the Snap4City Development model and CSBL approach on Dashboards and views. This approach allows to implement much smarter and dynamic business intelligence tools and smart applications.

1.4 – BEFDIT (Behavior Execution Framework for Digital Twins).

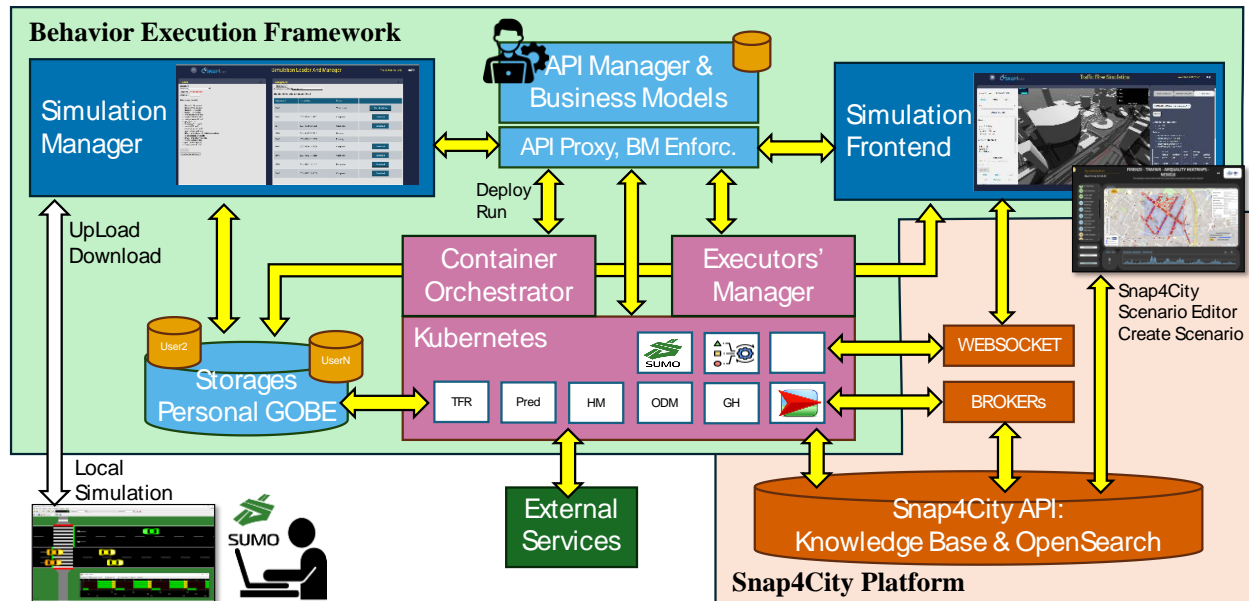
DAP and Simulation is a methodological approach for the analysis, design, and optimization of complex systems in domains such as industrial process design and urban mobility planning.

The Snap4City BEFDIT (**Behavior Execution Framework for Digital Twins**) framework addresses key limitations of most of complex digital twin behavioral supports through the following contributions by providing:

- **modular architecture** to support multiple DAP and simulation tools through a range of interfaces (e.g., API, WS, files), enabling co-simulation and co-execution of different tools which can contribute to the same digital twin behavior. It is a way to ensure seamless integration of heterogeneous behavioral components embedded in simulation scenarios, enabling dynamic interactions, agent-level intelligence, and feedback mechanisms across different simulation scales.
- **support for the integration** on the front-end of the simulations / DAP executors in “real time” and off-line via Web Interface. This point and previous are enabling solutions for co-working among experts on different fields/simulations models.
- **support for the scalable management** of the behavioral executions of DAP according to different business models in the range of “as a service” approach based on cloud/containers. Thus, opening the space to add other executors/simulations in the framework, supporting both open source and proprietary solutions. This is driver for reducing costs.

BEFDIT support the concept of **Behavioral Executors, BEX, or DAPs** are capable of moving vehicle agents,

computing routing for each vehicle, computing pollutant diffusion, making traffic flow predictions, computing emissions and costs, computing thermal evolution in building, computing the energy produced by irradiation, etc. Groups of **BEX** can be activated on demand.



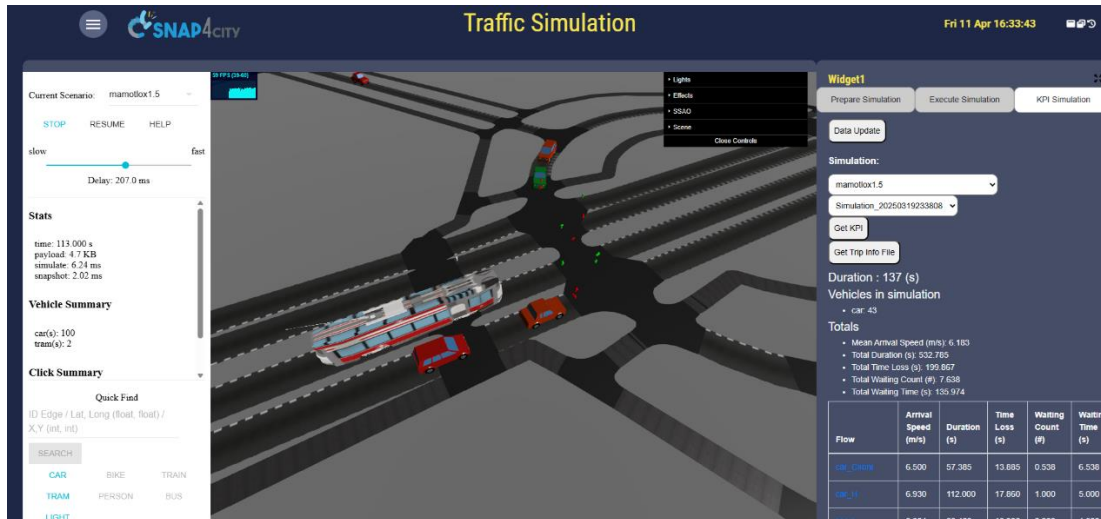
The solution enables users to configure and execute digital twins taking into account the behavioral aspects in the mobility and transport domain and it has been developed to extend the Snap4City open-source platform. **BEFDIT** consists of the following main components.

Simulation Manager, which manages the data regarding simulation as they are the setup and results. It provides a user interface to allow the upload of files coming from local preparation of simulation and of download of results.

Kubernetes, Container Orchestrator and Executors' Manager (to manage BEXs). This area provides support on managing containers on which any kind of simulator and executor (BEX) can be deployed and run (in our cases: SUMO, traffic flow reconstruction, routing engines, predictions, etc). Each of them, may exploit services such as: those provide by Snap4City API, any other **External Services** via any protocol or API, IoT/WoT (internet/web of things) brokers, Websocket (the latter two: to provide real time data/updates from/to the front- and back-end); and may access to the user's personal storage.

API Manager & Business Models provide support for defining which API can be exploited as well as by front end simulations and tools, according to a range of business models.

Simulation Frontends is an exemplification of a large range of Snap4City Dashboards/Tools which can be built by exploiting API via Client-Side Business Logic in JavaScript, CSBL. Among the simpler front-end tools of Snap4City: the **Scenario Editor** of Snap4City which allows to extract/change a portion of the Digital Twin and save it on Snap4City storage, the **MultiDataMap** for rendering on maps/orthomaps: traffic, ODM, heatmaps, trajectories, sensor data, flows, vector fields, etc.; **Synoptic** frames in SVG; **3D rendering** tools for cities; **BIM viewers**; **Traffic Light Planer**; **Waste Management**, **Smart Light** management; **eShare** for car sharing; etc.



For example, BEFDIT allows the integration of simulated traffic data into the solution making it directly usable by existing tools for visualization, analytics, event management, or as input for new simulation and decision-support systems (see **Figure**). The use of simulated sensors enables several applications, including the validation of predictive models, testing of smart city services in controlled environments, provision of fallback data in case of real sensor failures, and support for digital twin configurations in urban mobility scenarios.

The BEFDIT is a scalable, modular, and interoperable environment designed to address the increasing complexity and fragmentation of simulation tools within smart city digital twin infrastructures, particularly for mobility and transport domains. BEFDIT successfully enables the integration and co-execution of heterogeneous behavioral models, supporting agent-based, discrete-event, continuous, and hybrid simulations, across multiple spatial and temporal scales. The framework's key contributions include: interoperability across diverse simulation tools through API-based communication and containerized deployments; Behavior execution as a service, leveraging business model integration for scalable and cost-effective simulation management; Real-time and offline co-simulation capabilities, supporting both human-driven what-if analyses and AI-driven optimization processes; Persistent, versioned simulation management, enabling reproducibility, scenario evolution, and extensive experiment tracking. BEFDIT's integration within the open-source Snap4City platform and its deployment in operational smart city contexts demonstrate its practical effectiveness. Use cases involving dynamic congestion management, real-time data generation, multi-simulator co-execution, and high-performance simulation orchestration highlight its adaptability, scalability, and robustness. Experiments and operational data collected over large-scale urban environments (e.g., Florence, Helsinki, Sweden, Greece) confirmed BEFDIT's capacity to handle millions of road elements and thousands of behavioral executions, with dynamic orchestration of cloud resources via Kubernetes clusters. These results validate the framework's capability to enhance smart city decision-support systems by providing tactical and strategic planning tools based on automated, interoperable, and scalable digital twin simulations. Future developments will focus on expanding the ecosystem of integrated simulators and behavioral executors, strengthening AI-driven optimization loops, and further automating multi-domain what-if analysis processes. Additionally, efforts will be made to enhance multi-user collaboration features, privacy-preserving simulations, and broader exploitation across national and international smart city initiatives. This result has been produced for the SASUAM scalability project, and for the OPTIFaaS Flagship project of CN MOST, the National Center on Sustainable Mobility in Italy (<https://www.centronazionalemost.it/>), and Snap4City as official infrastructure framework on which the experiments were conducted (<https://www.snap4city.org>). Snap4City is an open-source technology of DISIT lab.

This tool is accessible on Snap4City platform and some of its instances.

- This page: <https://www.snap4city.org/1014>

- **SASUAM:** Solutions for Safe, Sustainable and Accessible Urban Mobility
 - <https://www.snap4city.org/999>
- **OPTIFaaS:** Operation and Plan, Transport Infrastructure and Facilities Support as a Service
 - <https://www.snap4city.org/1008>
- CN MOST: <https://www.centronazionalemost.it/>

1.5 – Creation of Smart Applications using ML/AI supports.

The design of the **User Interface** implies the design of dashboards/views to be developed. Snap4City Dashboards/Views are composed by several graphical widgets accessing to: data Storages, Server side Processing Logic (IoT App) data/nodes, External Services, Synoptics, and Brokers. Moreover, most of the widgets may host JavaScript code, exploiting its functionalities and receiving events from others, to enforce client-side business logic, CSBL. Those widgets and dashboards can be used to implement smart applications of any kind.

How to proceed: This phase is performed by identifying **Data Representation and Graphic User Interaction**:

This phase has to answer at questions such as:

- How many dashboards or view I need to create, how large they are, on which device they are shown?
- The user interface is only a Monitoring data from Storage?
- Who is going to access to those dashboards?
- How much interactive and dynamic the views/dashboards should be?
 - Do we need a menu to navigate on a number of connected Dashboards?
- Which kind of visual rendering is more adequate?
 - Which kind of user interface I have to provide to the users?
 - Which kind of graphic user interface your users would prefer?
 - Which kind of widget? The answer is easy since the preferred rendering tool for each Data Model has been defined.
- How many users are going to use the interfaces?
 - It is a scenario for Control Room or for understanding (such as a Business Intelligence tool to play with data and study)?
- Which Entity Instances have to be shown?
- The user interface has to provide data table for browsing on data? And in which order?

Passive and Active Dashboards/views

The Dashboards are composed by widgets. Each widget may represent several data and has a specific graphic representation and user interaction. Before stating the design of the user interface, you have to know the capabilities of the Snap4City Dashboards which are very wide providing almost any kind of widgets and graphic representation for your data, and relationships among them to create not only good representations but also a good interaction design, to specific what is going to happen interacting with the graphic elements and data on your user interface.

In Snap4City, there is a specific tutorial for the Dashboard development with several examples and the full list of capabilities in SLIDES: <https://www.snap4city.org/download/video/course/p2/>

Dashboard Widgets:

- are the main components of the Dashboards/views.
- can be created/edited from the Dashboard Builder, resized, placed, changed in color, etc.
- can be configured to perform a periodic refresh of their data recollecting them from storage/API.
- can be created/connected to Proc.Logic / IoT App.
- can be event driven, so that they are capable to update their data without forcing any refresh to their data.
- can collected interaction from the user to send them on Client-/Server-Side Business Logic.

- can be controlled by other widgets.
- can be controlled by the Proc.Logic / IoT App which can command the widget to show specific data from the storage, specific values, etc., and their combinations (Server-Side Business Logic).
- can be controlled by the Client-Side Business Logic, CSBL, in JavaScript coded in other on in the same widget to send/receive command to show specific data from the storage, specific values, etc., and their combinations, and also some computation, etc.
- can presents dynamically data on the basis of a parameter in the call itself via CSBL
- can exploit data analytics, ML, AI, and any processes provided via API, from CSBL.
- can open other dashboards
- etc.

The architecture of the Dashboard Builder is represented in the following Figure. The Dashboard Builder is composed by three main blocks: the Widget Collection, the Dashboard Wizard, and the Dashboard Editor (which includes the CSBL Editor).

The **Dashboard Editor** is used to create/modify dashboards (including their logic, visual analytics, what-if tools, etc.), by collecting and configuring Widgets and their relationships, sizing and placing them into dashboard canvas]. Each widget has a number of capabilities in presenting data, collecting data and interacting with users and protocols. The **Widget Collection** includes several ready-to-use widgets and custom widgets (that can be created for implementing new interactive graphic representations and Synoptics by using any SVG graphic editor). Each Widget is realized as an independent module which can: (i) present information to the user, (ii) get actions/interactions from the user, and (iii) interact back and forward with different channels. Channels are implemented as protocols and formats and allow to exploit storage systems (e.g., knowledge bases, relational DB, ODBC, JDBC, NoSQL API), any heterogeneous data sources, connection protocols such as HTTP/ HTTPs, API REST, WebSocket, IoT Brokers API, API related to ML/AI processes produce by some Data Analytics and from MLOps and thus which are running on some CPU/GPU cluster or server, etc. Therefore, widgets can work/react in an event driven way by Web sockets, and also access the historical data (time series) of sensors, maps, heatmaps, traffic flows, origin-destination matrices (ODMs), as well as query GIS servers (e.g., a GeoServer via WMS, WFS protocols). Such dashboard editing/creation is simplified by the **Dashboard Wizard**, by means of which users can create/connect dashboards in a few steps, exploiting pre-build templates. Moreover, the related wizard guides users in the selection of the most appropriate widgets for displaying the data of interest, or stating from the preferred widget to identify the data which can be used for populating it, or stating from the map to identify the data which are present in the area and the widgets for their rendering, etc. The Wizard assists users by reducing complexity, providing suggestions on finding combinations between data types (time series, vectors, array, maps, trajectories, heatmaps, origin destination, point of interest, typical trends, histograms, etc.), and graphic representations (trends, multi-trends, pie, donut, maps, chords, hierarchies, solar, dendrograms, single content, Italian flag, traffic flow, 3D building, etc.). Once the editing operation has been completed, users can save the related dashboard (with the possibility to delegate it or grant access to different users) and it is made available in the dashboard collection.

Moreover, with the aim of enabling developers in using the Dashboard Builder to create custom visual analytics, business intelligence, and what-if analysis tools, a flexible approach for modeling any business logic is provided with two different manners: Server-Side Business Logic (SSBL) and Client-Side Business Logic (CSBL). According to the SSBL approach, some graphic Widgets of dashboards have a counter part in the Node-RED nodes and thus are regarded as MicroServices which the Node-RED can send data and controls to, and which the Node-RED can receive events/actions from, as provided by users. This approach allows the dashboard designer to create SSBL by using the visual programming in Node-RED. This approach also implies that once a new widget node is deployed on a Node-RED flow, the related widget is automatically created into the selected dashboard and a WebSocket secure connection is established. The integration of Dashboards with Node-RED is also used to activate Data Analytics (data processing with machine learning and artificial intelligence algorithms) based on user actions on dashboards and/or scheduling in Node-RED. The CSBL approach is realized by coding segments of JavaScript directly into the graphic interface configuration of widgets (green block in Figure).

The CSBL code can call: (i) any external APIs (purple blocks and arrows in Figure), (ii) any API and data base services of the Snap4City platform (blue blocks and arrows in Figure), API related to ML/AI processes produce by some Data Analytics and from MLOps and thus which are running on some CPU/GPU cluster or server, and (iii) specific functions to send/receive commands and data to other widgets (green block in Figure). This approach allows users who can interact with some widget graphic element (a line, a legend, a bar, a pin on map, etc.) to activate a rendering, a computing, or a visualization on one or more widgets in the dashboard, and even open another dashboard with some parameters. With a minimal JavaScript programming capability to code the logic in these dashboards, a user can add intelligence functionalities to any widget to retrieve data directly from internal and external sources and generate and catch messages from other widgets in an event-driven way.

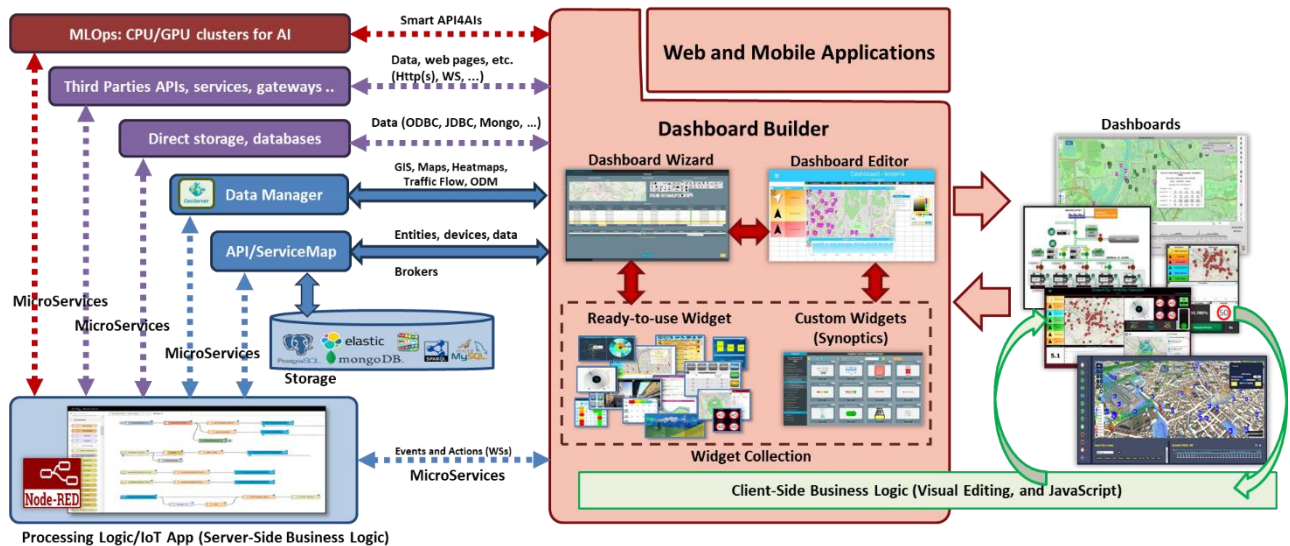


Figure – Dashboards / views and their connections with the platform and other services.

The dashboards can be classified into

- **Passive Dashboards:** showing data taken from **Storage** only, no actions toward Processing Logic (IoT App) node-RED neither on custom JavaScript (ONLY BLUE ARROWS in the above figure):
 - Passive dashboards may have widgets of any kind, and a lot of visualization tools without changing the status of Entities on platform, nor sending commands to the Server Side.
 - Passive dashboards are used to creates rendering views of the data in the storage and event driven from their changes with some limited logic pre-coded in the dashboard. For example, a Dashboard with a map and a menu from which the user may decide what is going to be visualized in the page, to browse the data, and see the historical time trend of the time series, etc.
 - <https://www.snap4city.org/download/video/course/p2/>
- **Active Dashboards,** are those that show that from the storage and in addition send/receive commands to/from the logic coded somehow (BLUE and GREEN ARROWS in the above figure) and in particular for
 - **Server-Side Business Logic** → logic on Processing Logic (IoT Apps) with Snap4City Dashboard Nodes, which is easier to be programmed begin based on Node-RED visual programming.
 - <https://www.snap4city.org/download/video/course/p2/>
 - <https://www.snap4city.org/download/video/course/p3/>
 - **Client-Side Business Logic** → logic on JavaScript on specific Dashboard Widgets only for authorized Area Managers developers of Snap4City Platforms. We suggest first prototype by using Server-Side Business Logic, then pass to Client-Side Business Logic in JavaScript. Client-Side Business Logic is coded into the Widgets providing events via the Dashboard Builder in Editing Mode. Client-Side Business Logic may exploit a large number of events provoked by the user on the Dashboard Widgets. **See development manual for CSBL:**

- <https://www.snap4city.org/download/video/ClientSideBusinessLogic-WidgetManual.pdf>
- **Third party APIs, services, gateways, pages** accessed by CSBL via some API to fill the Dashboards/views.
- **Third party databases** accessed by CSBL via some API to fill the Dashboards/views.
- Both kinds of Business Logics may be active on the same Active Dashboard.

The Active Dashboards are used to implement Business Intelligence solutions with high interactivity and the possibility of changing the data and the representation of data on the Dashboard/View dynamically on the basis of the user actions. Examples are:

- the click on some button or widget on Dashboard to activate a computation on Client/Server side. The computation on server side can include the activation of complex Data Analytics to be shown as a result on the same or other dashboard as event drive actions.
- To select/collect some data and perform a query showing them on map and barseries, pie, multi-trend, etc.
- To move a slider and see the light on dashboard changing.
- To click on a widget and activate / send control commands to other widgets and maps to drill down, drill up, zoom, on data, maps, etc.
- To filter data from a certain time windows and see the changes also on other data representations.
- Select a time period on a time series and see all the other time trends aligned to the same period.
- Select a PIN on map and see a barseries proving last year average data regarding that element.
- To invoke Data Analytics via some API.
- To include HTML/CSS custom widget with CSBL
- Etc.

A **dashboard** is substantially a view a tool to be used in both **Operation and Plan**, since it may include:

- **operational services** such as: monitoring data values, trends, events, alarms, conditions, etc., but also providing predictions, early warning, etc.
- **Decision Support System** tool, since it may provide evidence of normal and critical conditions, and in some cases may offer solutions, if well designed and connected with Data Analytics tools.
- **What-if analysis: a tool** to understand what is going to happen if something has been or is going to be changed (in traffic, road infrastructure for example), providing the evidence by prediction and simulation of what would be the effect of changes on again on Traffic but also emissions, travel time, congestion, etc. (can be used on Operation and Plan)
- **Simulation is a class of tools** to simulate the city conditions on the basis of changes: on traffic flow in specific parts, in road graph setting (adding/changing a road parameter as lanes, direction, position, velocity, etc.) (can be used in Operation and Plan)
- **Optimization is a class of tools** which are capable on the basis of an initial scenario and a set of parameters to provide you one or more possible solutions to solve a problem. For example, reduce congestion, reduce emissions and pollution, reduce the travel time, reduce the number of stops at the traffic lights, etc.
- **Scenarios is a class of tools** to select an area of the map and define a context including: road graph perimeter, eventual changes on the road graph, traffic in/out flux, included sensors of any kinds, etc.

2. MLOps in ClearML

2.1 – Managing an Experiment

In this section, we describe how to get started tracking experiments with ClearML.

2.1.1 – Credentials

The credentials to use ClearML MLOps in Snap4City are provided by RootAdmin of the Snap4city platform you are using.

In order for Jupyter notebooks or Python SDK to communicate with ClearML Server, you need to insert the credentials that can be generated by connecting to the ClearML graphical interface from the Snap4City interface and going to settings->workspace, these credentials are personal, in this way ClearML can connect the experiment to the user.

2.1.2 – Initialization of an Experiment

In order to connect to JupyterLab of Snap4City

Create a new notebook, by entering the credentials generated previously for example:

```
%env CLEARML_WEB_HOST=http://192.168.1.XXX:8080
%env CLEARML_API_HOST=http://192.168.1.XXX:8008
%env CLEARML_FILES_HOST=http://192.168.1.XXX:8081
# ClearML API Access Key and Secret
%env CLEARML_API_ACCESS_KEY=xxxx
%env CLEARML_API_SECRET_KEY=xxxxxx
```

2.1.3 – ClearML import

Install with pip all the packages needed for the experiment, even if the code will be executed by an agent on a different server than those available for Execution, it is necessary to communicate to ClearML server which packages must be used to run the experiment.

```
from clearml import Task
```

2.1.4 – Task Creation/Init

```
task = Task.init(project_name="project_name", task_name="task_name")
```

A dictionary with the parameters is created, so that they can also be managed later from a graphical interface, for example:

```
params = {'epochs': 1} task.connect(params)
```

The experiment is performed remotely by an agent:

```
task.execute_remotely(queue_name="queue_name")
```

At this point the local execution (in the JupyterLab) is stopped, and the experiment is queued to be executed by one of the agents.

By connecting to the ClearML graphical interface on Snap4City it is possible to monitor the experiment.

2.2 - Python Version and Package compatibilities in ClearML Agents

Each Server CPU/GPU task to be executed from the queue creates a Docker container using the preconfigured image in the agent as a base image (or another image if explicitly specified in the task). The process includes reading the requirements of the Python packages and their specific versions from the task, followed by installing and executing the task inside the container.

2.2.1 – Problems of compatibility among versions e packets Python

The communication of Python versions and required packages is done by the ClearML Server, based on the local development environment from which the task was initiated. For example, if you are using Python 3.10 and NumPy 1.26 in a JupyterLab environment and you initialize a task with Task.init(), the ClearML Server will record these requirements, and the agent will then try to satisfy them. However, if for example the Docker image used by the agent is based on Ubuntu 20.04, which supports Python 3.8 by default, a compatibility issue may arise. Python 3.8 does not satisfy the requirement for NumPy 1.26, which requires Python ≥ 3.9 , thus causing the task to fail due to version incompatibility.

A possible solution is to use the usual Python version both locally and in the Docker container. Alternatively, if the task fails, you can use the ClearML GUI to clone the task and change the versions of the required packages to be compatible with the container's Python version. You can also change the base Docker image used by the agent from the cloned task GUI.

INSTALLED PACKAGES

PIP

Q

📄

CLEAR

EDIT

```

aiohttp==3.9.3
aiosignal==1.3.1
async-timeout==4.0.3
attrs==22.2.0
certifi==2024.2.2
charset-normalizer==3.3.2
clearml==1.14.4
cmake==3.28.3
cycler==0.12.1
Cython==3.0.9
datasets==2.18.0
dill==0.3.6

```

CONTAINER

IMAGE

nvidia/cuda:11.0.3-cudnn8-runtime-ubuntu20.04

ARGUMENTS

SETUP SHELL SCRIPT

No changes logged

INSTALLED PACKAGES

PIP

```

aiohttp==3.9.3
aiosignal==1.3.1
async-timeout==4.0.3
attrs==22.2.0
certifi==2024.2.2
charset-normalizer==3.3.2
clearml==1.14.4
cmake==3.28.3
cycler==0.12.1
Cython==3.0.9
datasets==2.18.0
dill==0.3.6

```

CONTAINER

IMAGE

nvidia/cuda:11.0.3-cudnn8-runtime-ubuntu20.04

ARGUMENTS

SETUP SHELL SCRIPT

No changes logged

2.2.2 – Selection of Docker Image of the Agent for CPU Compatibility

In order for the agent to use the GPU correctly, it is essential to select a Docker image that guarantees compatibility with the CUDA drivers of the GPU in use. For example, for a machine equipped with an Nvidia 4090 GPU with CUDA driver version 12.3, an appropriate Docker image would be `nvidia/cuda:12.3.2-cudnn9-runtime-ubuntu22.04`.

This image not only supports the required CUDA driver version, but is also based on Ubuntu 22.04, which uses Python 3.10 by default.

<https://hub.docker.com/r/nvidia/cuda/tags>

2.3 - ClearML Basic Functionalities

2.3.1 - Log of the Hyper-parameters

Saving hyperparameters for each experiment allows to replay the experiment with different parameters and compare parameters from multiple experiments.

You can save parameters as:

- dictionary (very useful when parsing an external configuration file and storing it as a dict object),
- configuration files,
- custom objects, or
- Hydra configurations.

```
params_dictionary = {'epochs': 3, 'lr': 0.4}
task.connect(params_dictionary)
```

2.3.2 - Log Artefacts

ClearML allows you to store the outputs of your experiments as artifacts, which can then be easily accessed and used, either through the web interface or programmatically.

ClearML provides methods to easily track the files generated during the execution of your experiments, such as:

- Numpy objects
- Pandas DataFrames
- PIL (Python Imaging Library)
- Files and folders
- Python objects
- Other

2.3.3 - Add Artefacts

Load a file

```
task.upload_artifact(name='data',
artifact_object='/path/to/preprocess_data.csv')
```

Load a folder

```
task.upload_artifact(name='folder', artifact_object='/path/to/folder/')
```

Load an instance of an object, corresponding formats for Numpy/Pandas/PIL Images such as `npz/csv.gz/jpg` are supported. If the object type is unknown, ClearML serializes (pickles) it and loads the pickle file.

```
numpy_object = np.eye(100, 100)
task.upload_artifact(name='features', artifact_object=numpy_object)
```

2.3.4 – Use the Artefacts

Registered artifacts can be used by other tasks. To use an artifact, you must first get an instance of the task that originally created it, then you can download it and get its path, or get the artifact object directly.

```
preprocess_task = Task.get_task(task_id='preprocessing_task_id')
local_csv = preprocess_task.artifacts['data'].get_local_copy()
```

2.3.5 - Models

Models are a special type of artifact. Models created by popular frameworks (such as PyTorch, TensorFlow, Scikit-learn) are automatically registered by ClearML. All snapshots are automatically registered.

2.3.6 – Load of a model

Loading a trained model is similar to loading an artifact

```
prev_task = Task.get_task(task_id='the_training_task')
last_snapshot = prev_task.models['output'][-1]
local_weights_path = last_snapshot.get_local_copy()
```

2.3.7 - Log Metrics

By default, ClearML automatically captures and logs everything that is reported to TensorBoard and Matplotlib. Since not all metrics are plotted this way, you can also manually report metrics using a **Logger** object. You can log time series data and confusion matrices, HTML, audio and video, custom plotly graphs, etc.



2.3.8 – Data types registered with Logger

- **Text:** Mostly captured automatically from stdout and stderr, but can be logged manually.
- **Scalars:** Time series data. The X-axis is always a sequential number, usually iterations but can be epochs or others.
- **Graphs:** General graphs and charts, such as histograms, confusion matrices, line plots, and custom plotly graphs.

- **Debug Samples:** Images, audio, and video. Can be flagged per iteration.



2.3.9 – Automatic Recording of metrics

ClearML automatically captures metrics reported to major libraries, such as TensorBoard and Matplotlib, without the need for additional code.

In addition, ClearML captures and logs everything written to the console, from debug messages to errors to library warning messages.

GPU, CPU, Memory, and Network information are also automatically captured.



2.3.10 - Manual Recording of Metrics

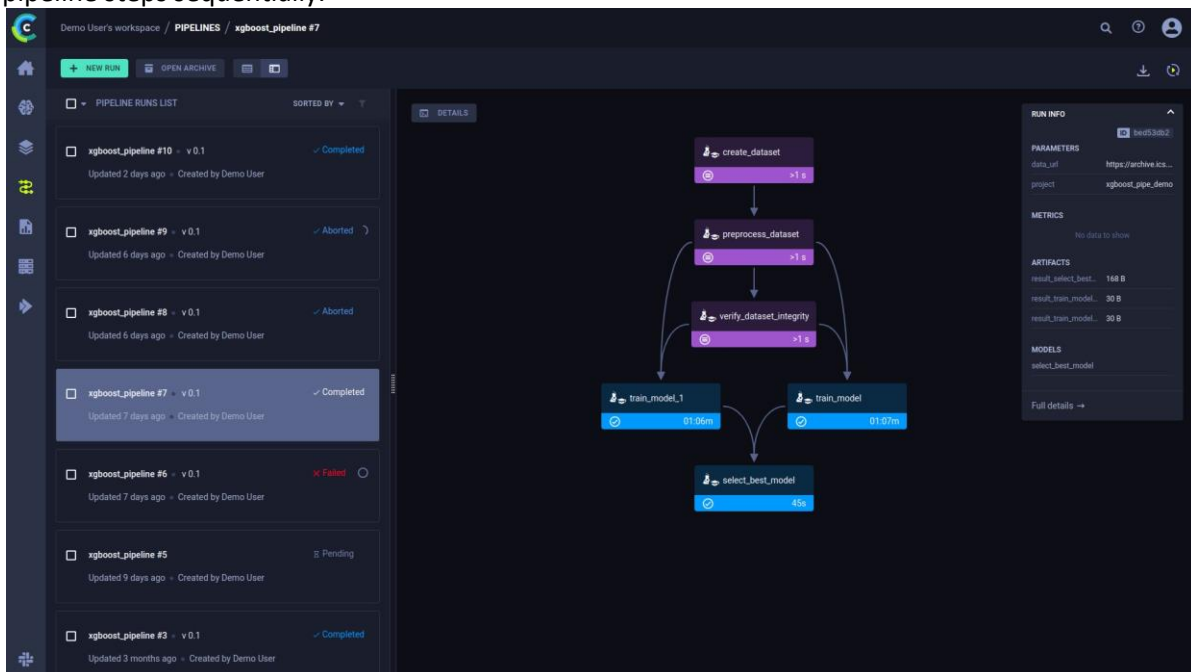
ClearML also supports manual reporting of different types of metrics and graphs, such as line charts, histograms, and even Plotly graphs. The object used to report metrics is called a logger and is obtained by calling `Task.get_logger()`.

- Text
- Scalars
- Graphs
 - 2D Graphs
 - Histograms
 - Confusion Matrices
 - Scatter Plots
 - 3D Graphs
 - Surface Plots
 - Scatter Plots
 - Tables
 - Pandas DataFrames

- CSV Files
 - Matplotlib Figures
 - Plotly Figures
- Debug Samples
 - Images
 - HTML
 - Media - Images, Audio, Video

2.3.11 - Pipelines

Pipelines are a way to connect multiple processes, using the output of one process as input to another. ClearML Pipelines are implemented by a Task Controller that holds the logic of the interactions between the pipeline steps. The execution logic controls which step to launch based on the completion of the parent steps. Depending on the specifications defined in the task controller, the parameters of a step can be overridden, allowing users to leverage the products of the execution of other steps, such as artifacts and parameters. When executed, the controller will launch the pipeline steps sequentially.



The pipeline execution page in the web interface shows the pipeline structure in terms of the steps executed and their status, as well as configuration parameters and execution output.

ClearML pipelines are created from code using one of the following methods:

- **PipelineController** class: A python interface to define and configure the pipeline controller and its steps. The controller and steps can be functions in your python code, or existing ClearML tasks.
- **PipelineDecorator** class: A set of Python decorators that transform your functions into the pipeline controller and steps.

When the pipeline is executed, tasks are created for the controller and steps.

Since a controller is itself a ClearML task, it can be used as a pipeline step. This allows you to create more complex workflows, such as pipelines that execute other pipelines, or pipelines that execute multiple tasks at the same time.

2.4 - ClearML Data

ClearML Data is a key component of ClearML that provides advanced versioning of datasets, simplifying access and traceability of data used in machine learning experiments. This tool is essential for ensuring the reproducibility of experiments and streamlining the workflow in machine learning, especially when datasets are subject to continuous updates.

ClearML Data is based on two main goals (ClearML Data, 2024):

- **Accessibility:** Ensure that data is easily accessible from any machine connected to the system.
- **Versioning:** Link data to experiments, providing complete and precise traceability of the versions of datasets used in the various tasks.

These features allow developers to track the evolution of datasets over time and easily access specific versions for experiment reproduction and analysis.

2.4.1 - Dataset Versioning

One of the most powerful aspects of ClearML Data is its ability to version datasets. Each dataset can be finalized once it is complete, preventing further changes, which ensures that each experiment uses exactly the data intended without the risk of accidental changes. When a dataset needs to be updated, ClearML can create a new version that inherits the contents of the previous dataset, maintaining a lineage of the data.

This approach optimizes storage space, as ClearML only stores the differences between dataset versions using differential storage. This is extremely useful in cases where a project requires multiple updates to datasets without having to duplicate the entire data for each version.

2.4.2 - Dataset Organization and Access

ClearML Data allows you to organize datasets into projects and subprojects, making them easy to manage and search. Datasets can be further enriched with tags, which makes it easy to retrieve the most recent or relevant versions using commands like `Dataset.get()` within a specific project or tag. This makes the workflow much more efficient, as datasets are automatically retrieved and managed, avoiding errors due to versions or outdated files.

As with any task in ClearML, datasets can be organized into projects (and subprojects). Additionally, when creating a dataset, you can apply tags to the dataset, which will make it easier to search.

Organizing your datasets into projects by use case makes it easier to access the latest version of the dataset. If only one project is specified when using `Dataset.get()`, the method returns the latest dataset in a project. The same goes for tags; if a tag is specified, the method will return the latest dataset labeled with that tag.

In cases where you are using a dataset in a task (for example, consuming a dataset), you can easily track which dataset the task is using by using the `alias` parameter of `Dataset.get()`. Pass `alias=<dataset_alias_string>`, and the task using the dataset will store the dataset ID in the `dataset_alias_string` parameter under the `CONFIGURATION > HYPERPARAMETERS > Datasets` section of the task.

2.4.3 - Dataset Documentation and Tracking

ClearML Data not only manages the content of datasets, but also provides tools to document and track additional details. Through the `Logger` object, users can attach metrics and debug samples directly to the dataset, adding information such as summary tables and statistics generated during the pre-processing or data ingestion process. This improves visibility into the dataset and makes it easier to understand the content and transformations applied.

Attach informative metrics or debug samples from the dataset directly to the `Dataset`. Use `Dataset.get_logger()` to access the dataset's logger object, then add any additional information to the dataset, using the methods available with a `Logger` object. You can add dataset summaries (such as tables) to create a preview of the stored data for better visibility, or attach any statistics generated by the data ingestion process.

2.4.4 - Using ClearML Data

ClearML Data offers two main interfaces for managing datasets:

- **CLI (Command Line Interface):** The `clearml-data` tool lets you create, load, add, remove, and manage datasets directly from the command line. It is especially useful for automating the process of versioning and managing data in continuous development contexts.
- **Python SDK:** The `clearml.Dataset` interface lets you manage datasets programmatically within Python scripts. Developers can create datasets, access specific versions, add or remove files, and get local copies of the data. The ability to integrate dataset management into pre- or post-processing workflows greatly improves the efficiency of the development cycle.

ClearML Data lets you efficiently synchronize changes to datasets, reflecting local updates on the server without the need to manually manage files. Using commands such as `sync_folder()`, developers can automatically update a dataset based on changes made to a specific local folder. This is especially useful when there is a single “point of truth” for the data, which is periodically updated, and you want these changes to be reflected in ClearML.

Datasets can be retrieved and used easily through the CLI or Python SDK. ClearML supports two access modes:

- **Read-only copy:** Ideal for maintaining an immutable version of the dataset.
- **Writable copy:** Provides a local, editable copy of the dataset, useful for testing or temporary data manipulation.

Additionally, ClearML allows you to add or remove files from an existing dataset, while still maintaining traceability of the changes. Once the changes are complete, you can upload them back to the server and finalize the dataset to make it available for future tasks.

ClearML provides an intuitive graphical interface that allows you to navigate through datasets and view the lineage of versions as shown in Figure 2-6, Figure 2-7, Figure 2-8, and Figure 2-9. Users can filter datasets by tag or creator, and each dataset version displays details about its content, including thumbnails, tables, and associated files. This feature simplifies data management and improves transparency and access to key information.

2.4.5 - ClearML Data CLI

create

Create a new dataset.

```
clearml-data create [-h] [--parents [PARENTS [PARENTS ...]]] [--project  
PROJECT] --name NAME [--version VERSION] [--output-uri OUTPUT_URI]  
[--tags [TAGS [TAGS ...]]]
```

add

add files and folders to dataset.

```
clearml-data add [-h] [--id ID] [--dataset-folder DATASET_FOLDER]  
[--files [FILES [FILES ...]]] [--wildcard [WILDCARD [WILDCARD ...]]]  
[--links [LINKS [LINKS ...]]] [--non-recursive] [--verbose]
```

remove

remove files from a dataset

```
clearml-data remove [-h] [--id ID] [--files [FILES [FILES ...]]]
[--non-recursive] [--verbose]
```

upload

Load changes in the dataset on server.

```
clearml-data upload [-h] [--id ID] [--storage STORAGE] [--chunk-size
CHUNK_SIZE] [--verbose]
```

close

Finalize the dataset and make it ready for use. This automatically uploads all files that were not previously uploaded. Once a dataset is finalized, it cannot be modified.

```
clearml-data close [-h] [--id ID] [--storage STORAGE] [--disable-upload]
[--chunk-size CHUNK_SIZE] [--verbose]
```

sync

Synchronize the contents of a folder with ClearML. This is useful if a user has a single point of truth (i.e. a folder) that is updated from time to time. When an update needs to be reflected in the ClearML system, call `clearml-data sync` and pass the path to the folder, and the changes (either adding, modifying, or removing files) will be reflected in ClearML. This command also loads the data and automatically finalizes the dataset.

```
clearml-data sync [-h] [--id ID] [--dataset-folder DATASET_FOLDER]
--folder FOLDER [--parents [PARENTS [PARENTS ...]]] [--project PROJECT]
[--name NAME][--version VERSION] [--output-uri OUTPUT_URI] [--tags [TAGS
[TAGS ...]]][--storage STORAGE] [--skip-close] [--chunk-size CHUNK_SIZE]
[--verbose]
```

get

Get a local copy of a dataset. By default, you get a read-only cached folder, but you can get a modifiable copy using the `--copy` flag.

```
clearml-data get [-h] [--id ID] [--copy COPY] [--link LINK] [--part
PART] [--num-parts NUM_PARTS] [--overwrite] [--verbose]
```

2.4.6 - ClearML Data SDK

Per cominciare Importa in Python la classe Dataset.

```
from clearml import Dataset
```

Create Dataset

ClearML Data supports multiple ways to programmatically create datasets:

- `Dataset.create()`: Creates a new dataset. Parent datasets can be specified, from which the new dataset will inherit its data.
- `Dataset.squash()`: Generates a new dataset by merging a set of related datasets together.

You can add metadata to your datasets using `Dataset.set_metadata()`, and access the metadata using `Dataset.get_metadata()`.

`Dataset.create()`

Use the `Dataset.create` method to create a dataset. Creating datasets programmatically is especially useful when preprocessing data, so that the preprocessing code and the resulting dataset are stored in a single task (see the `use_current_task` parameter in `Dataset.create`).

```
# Preprocessing code here
dataset = Dataset.create(
    dataset_name='dataset name',
    dataset_project='dataset project',
    parent_datasets=[PARENT_DS_ID_1, PARENT_DS_ID_2],
    dataset_version="1.0",
    output_uri="gs://bucket-name/folder",
    description='my dataset description')
```

Access to Datasets

Once a dataset has been created and uploaded to a server, the dataset can be accessed programmatically from anywhere.

`Dataset.create()`

Use the `Dataset.get` class method to access a specific Dataset object, providing any of the following dataset attributes: dataset ID, project, name, tags, and/or version. If multiple datasets match the query, the most recent one is returned.

```
dataset = Dataset.get(
    dataset_id=None,
    dataset_project="Example Project",
    dataset_name="Example Dataset",
    dataset_tags="my tag",
    dataset_version="1.2",
    only_completed=True,
    only_published=False,
)
```

Once you have a specific dataset object, you can get a local copy of the dataset using one of the following options:

- `Dataset.get_local_copy()`: Get a local, read-only copy of an entire dataset. This method returns a path to the dataset in the local cache (downloading the dataset if it is not already cached).
- `Dataset.get_mutable_local_copy()`: Get a local, writable copy of an entire dataset. This method downloads the dataset to a specific (uncached) folder, specified with the `target_folder` parameter. If the specified folder already has content, specify whether to overwrite its contents with the dataset's contents, using the `overwrite` parameter.

Modify a Dataset

Once you have created a dataset, you can modify and replace its contents. As your data changes, you can add updated files or remove unnecessary files.

`add_files()`

To add local files or folders to the current dataset, use the `Dataset.add_files` method. If a file is already in a dataset, but has changed, it can be added again and ClearML will load the file diff.

`remove_files()`

To remove files from a current dataset, use `Dataset.remove_files()`. Enter the path to the folder or file to remove in the `dataset_path` parameter. The path is relative to the dataset.

You can enter a wildcard in `dataset_path` to remove a set of files that match the wildcard. Set the recursive parameter to `True` to match all files in the wildcard recursively.

```
dataset.remove_files(dataset_path="*.csv", recursive=True)
```

Preview of Datasets

Add informative metrics, graphs, or averages to the Dataset. Use `Dataset.get_logger()` to access the dataset's logger object, then add any additional information to the dataset, using the methods available with a Logger object. You can add dataset summaries (such as tables) to preview the stored data, or attach any statistics generated by the data ingestion process.

```
# Attach a table to the dataset
dataset.get_logger().report_table(
    title="Raw Dataset Metadata", series="Raw Dataset Metadata",
    csv="path/to/csv"
)

# Attach a histogram to the table
dataset.get_logger().report_histogram(
    title="Class
distribution",
    series="Class
distribution",
    values=histogram_data,
    iteration=0,
    xlabel=histogram_data.index.tolist(),
    yaxis="Number of samples",)
```

Upload Files

To upload dataset files to storage, use the `Dataset.upload` method. The dataset files must be uploaded

before the dataset is finalized.

Finalize a Dataset

Use `Dataset.finalize()` to close the current dataset. This marks the dataset activity as Completed, after which the dataset can no longer be modified.

Before finalizing a dataset, its files must first be uploaded.

Synchronize Local Storage

Use `Dataset.sync_folder()` to update a dataset based on changes to the contents of a specific folder. Specify the folder to synchronize with the `local_path` parameter (the method takes all files within the folder and recursively).

This method is useful in the case where there is a single point of truth, be it a local or network folder, that is updated periodically. Changes to the folder will be reflected in a new version of the dataset. This method saves time since you don't have to manually update (add/remove) files in a dataset.

2.4.7 - Graphic User Interface

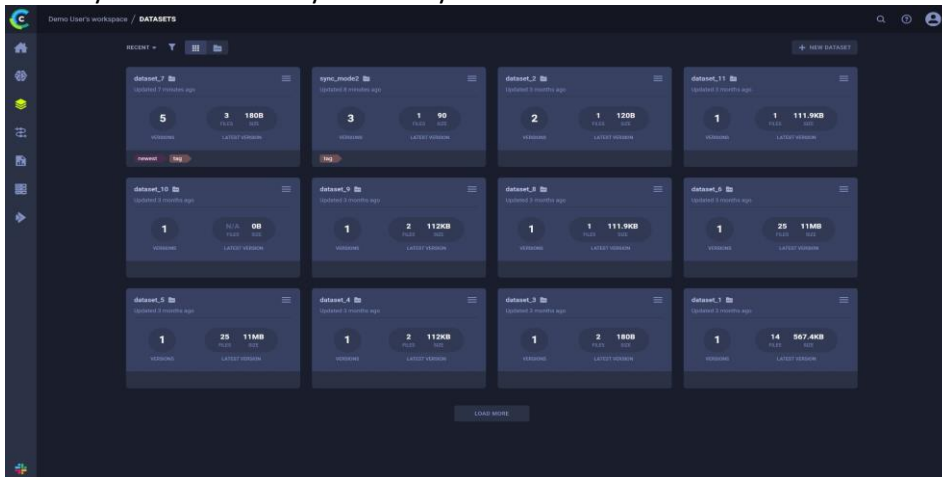
Use the Datasets page to navigate and manage datasets. The page displays all datasets created using ClearML Data.

You can view the datasets page in Project view or List view.

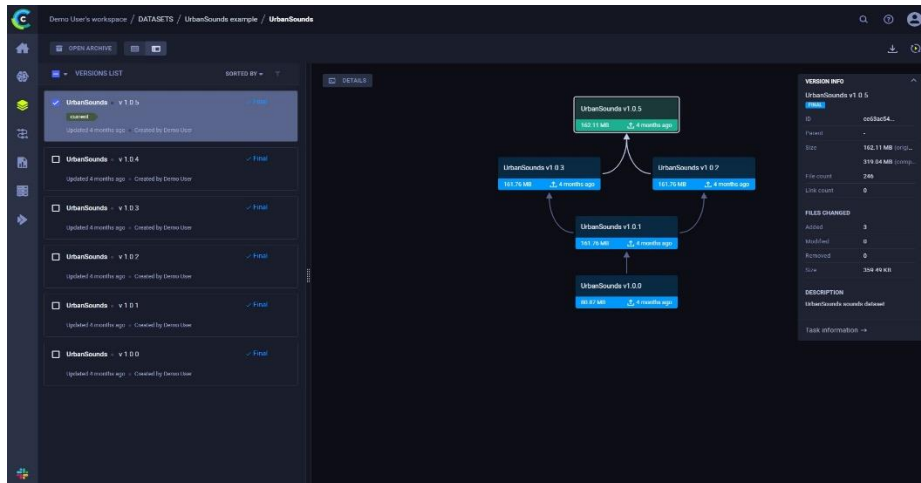
Click on a dataset tab to navigate to its Version List, where you can view the lineage and contents of the dataset's versions.

Filter datasets to more easily find what you're looking for. These filters can be applied by clicking Filter:

- My Work - Show only datasets you've created



- Tags - Choose which tags to filter from a list of tags used in the datasets.



The dataset page lists the versions of the dataset. Selecting a version causes the panel to display its lineage graphically.

Version Detail Panel

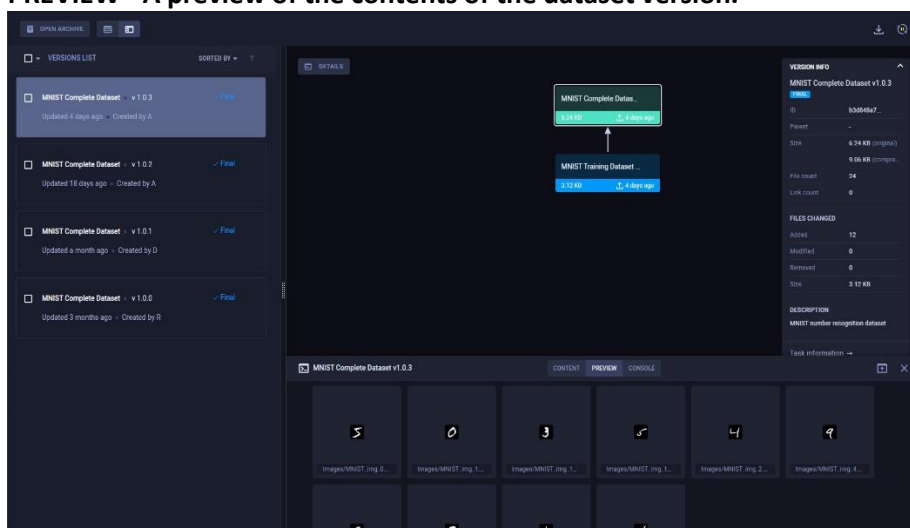
Click DETAILS in the upper left corner of the information panel to display the version details panel. The panel includes the tabs:

CONTENT - A table that summarizes the contents of the version, including file names, file sizes, and hashes.

The screenshot shows the 'MNIST Complete Dataset v1.0.3' version details panel. The 'CONTENT' tab is active, displaying a table of files. The table has columns for File Name, File Size, and Hash. The files listed are 'TEST/MNIST_img_0.png', 'TEST/MNIST_img_1.png', 'TEST/MNIST_img_10.png', 'TEST/MNIST_img_11.png', and 'TEST/MNIST_img_2.png'. The 'VERSION INFO' panel on the right shows the version details, including the ID, Parent, Size, File count, and a 'FILES CHANGED' section with a table of changes.

File Name (24 files)	File Size (total 6.24 KB)	Hash (SHA256)
TEST/MNIST_img_0.png	313 B	509F7696b0a186351d1fa4078a6f0970ba99fa79bdc773db8827
TEST/MNIST_img_1.png	312 B	5e1c0b44839fca50f70579a080a0a9877ba331ab0f547a991d6
TEST/MNIST_img_10.png	287 B	c9da43342aee7a375d4e379ba05a0813ba0b91c5d75b471e
TEST/MNIST_img_11.png	312 B	293d27b05371a2f05d444d41ba029295a67f40a8346d7d0d549
TEST/MNIST_img_2.png	289 B	356d45d9778a623a3e9133f89a610273a97a03395a64a43897

PREVIEW - A preview of the contents of the dataset version.



3. ClearML - Experiment Comparison

ClearML provides advanced tools for comparing machine learning experiments, giving researchers and developers a flexible and customizable platform to analyze results, compare hyperparameters, and optimize model performance. This functionality is essential for improving development cycle efficiency and monitoring model performance across various configurations.

3.1 - Experiment Table

The Experiment Table is a customizable list of experiments associated with a project. From the Experiment Table, you can view experiment details and work with them (reset, clone, queue, create leaderboards to monitor experimentation, and more). The auto-updating Experiment Table allows users to continuously monitor the progress of experiments.

The Experiment Table in ClearML can be customized in several ways:

- **Column order and size:** You can change the order of columns by dragging them and adjust their width.
- **Editing columns:** You can show or hide columns and add custom columns for metrics and hyperparameters.
- **Filter and sort columns:** Filter and sort columns based on various criteria, such as experiment type, metrics, hyperparameters, and more.

These customizations are useful for creating real-time leaderboards, sorting models by specific metrics, and tracking hyperparameters. Changes made are persistent and can be saved and shared via URL, making it easy for ClearML users to collaborate.

ClearML

Demo User's workspace / PROJECTS / Example Project

+ NEW EXPERIMENT

OPEN ARCHIVE

OVERVIEWEXPERIMENTSMODELS

3.2 - Experiments Leaderboard

To create a custom leaderboard of experiments in ClearML: Add experiment configurations: Include hyperparameters.

- **Edit and add properties to experiments:** Update or insert new properties.
- **Insert reported metrics:** Choose between the last reported value, minimum, or maximum of each time series metric.

- **Filter by user or experiment type:** Use specific filters to select experiments.
- **Use and filter by specific tags:** Set and apply filters based on tags.

Sort the table by performance metrics or other criteria, and also filter experiments by name. The custom dashboard can be shared via URL and bookmarked for future use.

Lapo Bartolacci's workspace / PROJECTS /

UrbanSounds

OVERVIEW

EXPERIMENTS

MODELS

SEARCH

REFRESH

SETTINGS

EXPORT

NEW EXPERIMENT

OPEN ARCHIVE

IMPORT

EXPORT

TAGS

STATUS

USER

ITERATION

Args.batch_size

Args.dropout

Args.number_of_epochs

training loss

EXAMPLE

16x batch

True

Completed

ClearML

20

0.005

64

0.32

10

0.985828

EXAMPLE

16x batch

Depth

Lat

Completed

ClearML

70

0.002

16

0.3

10

0.979977

EXAMPLE

4x batch

True

Completed

ClearML

260

0.005

4

0.32

10

0.945886

EXAMPLE

16x batch

True

Completed

ClearML

70

0.005

16

0.3

10

0.481912

EXAMPLE

original

Completed

ClearML

260

0.002

4

0.3

10

0.30538

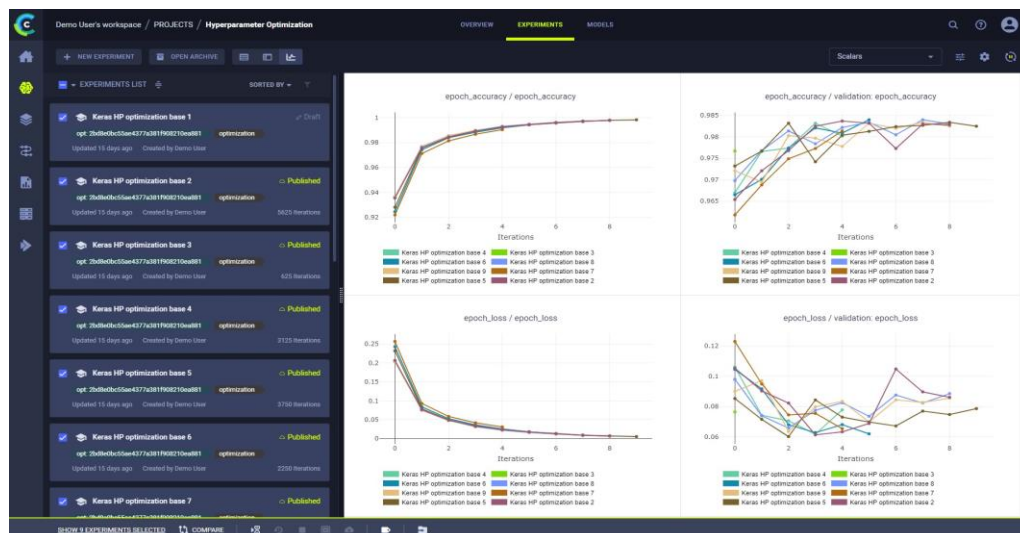
3.3 - Select Multiple Experiments

Select multiple experiments using the checkboxes to the left of the experiments.

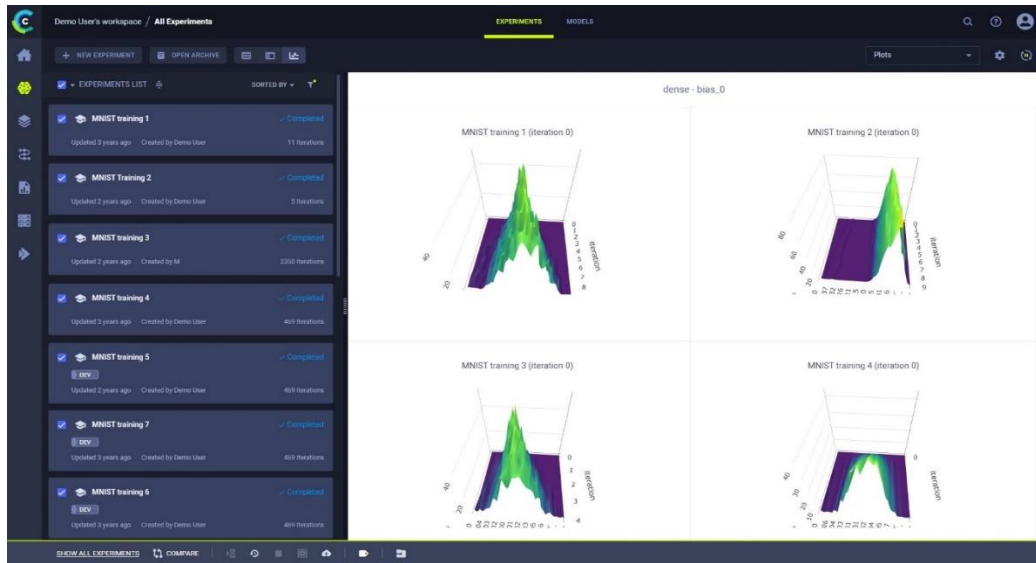
In the ClearML comparison view, you can compare the scalar results and plots of the selected experiments. If no specific experiments are selected, the view will automatically display all experiments visible in the table. In the drop-down menu, you can choose to display:

- **Scalars:** Line plots that represent the scalar results of the experiments as a time series.
- **Graphs:** The last sample reported for each metric/variant combination for each experiment being compared.

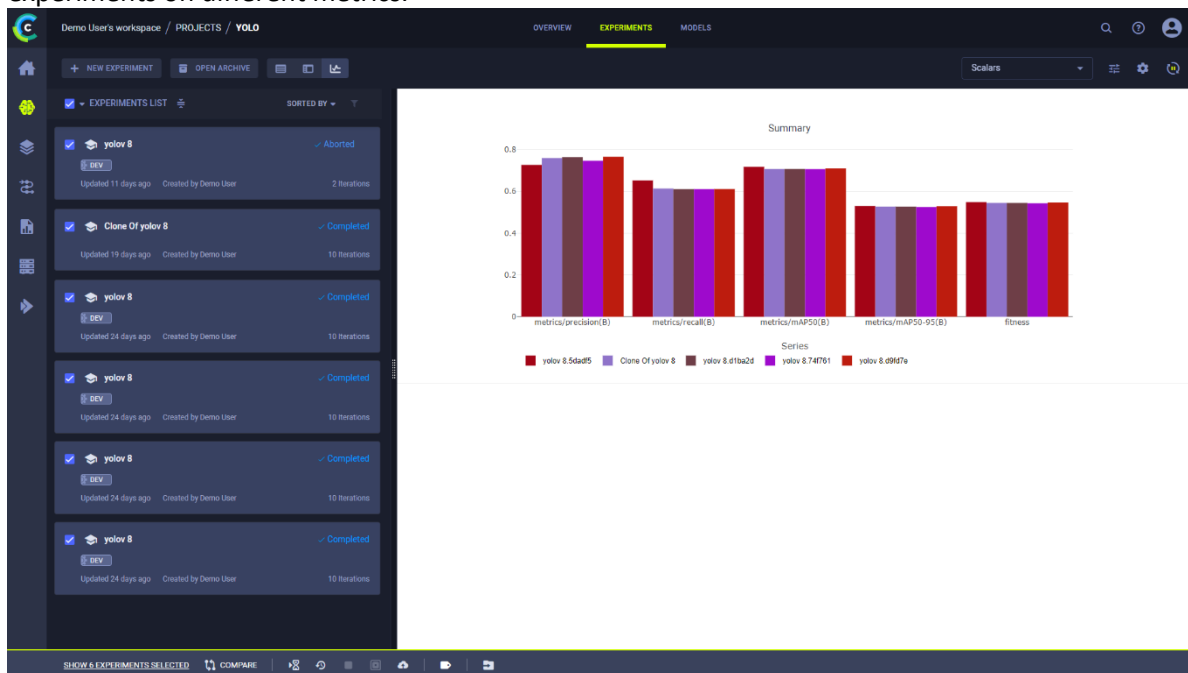
Line, scatter, and bar plots are compared by overlaying the metrics/variants of all experiments into a single comparison plot. This allows for a direct visual comparison between experiments.



Other graph types are displayed separately for each experiment.



In the ClearML comparison view, single-valued scalars from different experiments are displayed in a bar chart. Each group of bars in the chart represents a reported metric, with each bar within the group representing a specific experiment. This format makes it easy to directly compare the performance of experiments on different metrics.



3.4 - Detailed Comparison

Once multiple experiments are selected, you can click COMPARE to access a detailed page where you can compare the experiments.

3.4.1 - Comparison Modes

Side-by-Side Textual Comparison

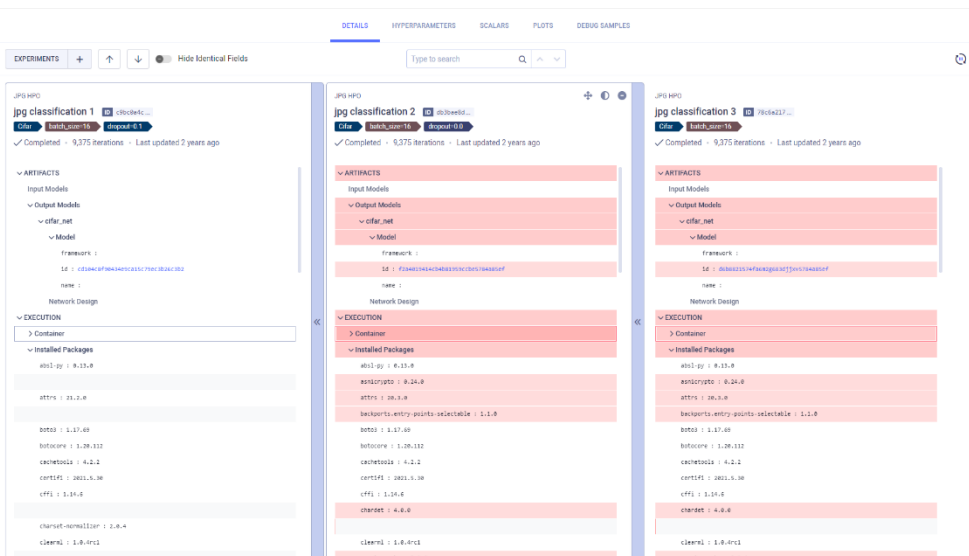
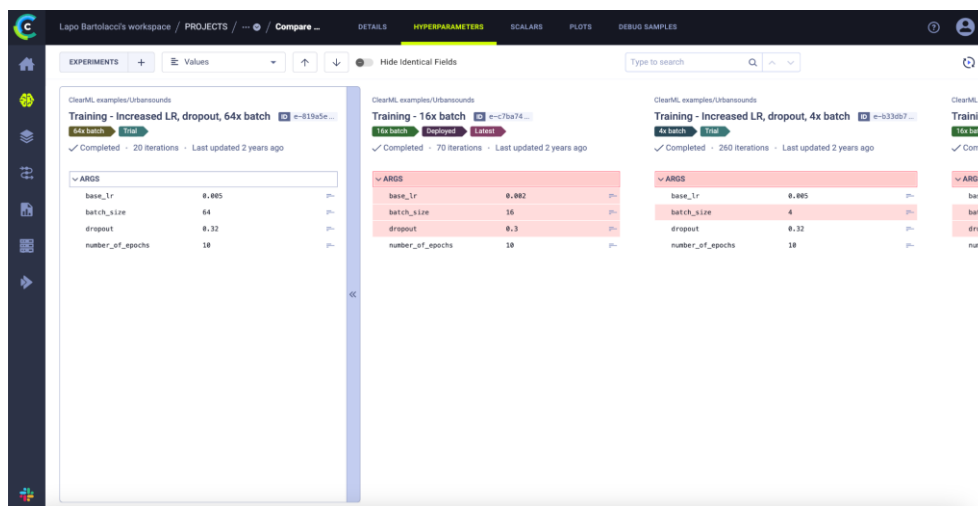
ClearML also offers a side-by-side textual comparison mode, which allows you to compare the details

and hyperparameters of experiments in detail:

- In the Details tab, you can compare aspects such as source code and Python packages used, as shown in Figure 2.15.
- In the Hyperparameters tab, users can compare nominal values of experiment configuration parameters, with the option to highlight differences between selected experiments.
- In the Scalars tab, you can compare values of different metrics, highlighting maximums and minimums, as shown in Figure 2.16.

Side-by-side comparison makes it easy to analyze differences between experiments, making it easier to optimize and debug models.

Experiments are arranged in vertical tabs for side-by-side comparison, with the experiment on the left serving as the base for comparison. You can change the base experiment and the differences between experiments are highlighted. You can navigate through the differences and search for specific fields or values, with the option to hide identical fields to focus on the differences.



Scalars Table Comparison

In the "Scalars" tab, the metric values of the experiments are organized in a table with metrics/variants per row and experiments per column.

You can choose to display the last values, minimum or maximum values reported during the

execution of the experiments via a drop-down menu.

You can download this table as a CSV file.

Additionally, there is an option to highlight the maximum and minimum values of each variant in the table.

		jpg classification 1	jpg classification 2	jpg classification 4	jpg classification 3	jpg classification 5
accuracy per class	horse	72.2 [MAX]	70.5 [MIN]	71.7	71.3	71.6
	car	72.2	70.3 [MIN]	74.1 [MAX]	74	73.3
	cat	40.5 [MAX]	35	37.7	33.4 [MIN]	34.3
	deer	44	37.2 [MIN]	47	53.4 [MAX]	45.3
	plane	63.1	59.5	64.9 [MAX]	57.2 [MIN]	62.4
	bird	33.9	41.2 [MAX]	39.5	33.3 [MIN]	41.1
	truck	62.4	63.1 [MAX]	59.2	57.6 [MIN]	61.7
	frog	74.6	75.5	70.6 [MIN]	77.1 [MAX]	73.9
	ship	61.2	80.2	76.9 [MIN]	81.3 [MAX]	79.4
	dog	49.9	58.9 [MAX]	47.3 [MIN]	52	52.2
accuracy	total	59.4	59.14	56.89 [MIN]	59.06	59.52 [MAX]
training loss	training loss	1.2447971	1.2363554 [MIN]	1.252396	1.2613677	1.270214 [MAX]

Parallel Coordinates Mode

The "Parallel Coordinates" mode in the "Hyperparameters" tab allows you to visualize the impact of hyperparameters on a specific metric. To do this:

1. Select a metric to compare under "Performance Metric".
2. Choose the metric values to use in the graph: "LAST" for the last value or the most recent value in the running experiments, "MIN" for the minimum value, "MAX" for the maximum value.
3. Select the hyperparameters to compare under "Parameters".



Comparing Plots

In the "Scalars" and "Plot" tabs, you can compare the plots of your experiments.

The "Scalars" tab displays scalar values as time-series line graphs, while the "Plot" tab compares the last reported sample of each metric/variant combination for each experiment.

Line, scatter, and bar plots are compared by overlaying each metric/variant from all experiments in a single comparison plot.

You can use the "Group by" option to choose how to group the plots: by "Metric" (all variants of a metric in the same plot) or by "Metric+Variant" (each variant in its own plot, the default).



UNIVERSITÀ
DEGLI STUDI
FIRENZE

DINFO
DIPARTIMENTO DI
INGEGNERIA
DELL'INFORMAZIONE

DISIT
DISTRIBUTED SYSTEMS
AND INTERNET
TECHNOLOGIES LAB



Smoothing **e**

Exponential Moving Ave... **•**

[Find series](#)

v accuracy "accuracyperclass"

frog

plane

ship

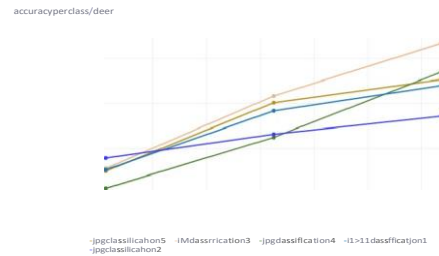
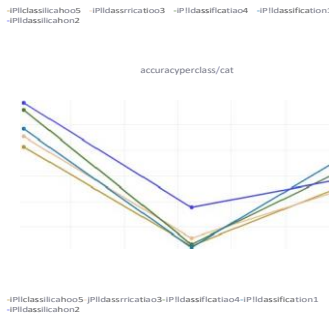
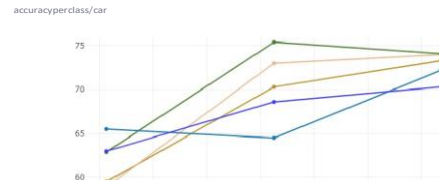
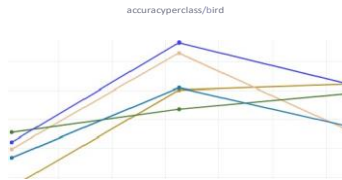
"trainingloss"

training loss

EXPERIMENTS **+**

Find plots

Q



dense-bias_O

dense-kernelO

dense_l-bias_O

dense_l-kernelO

dense_l-bias_O-histogram

dense_l-kernelO-histogram

dense_2-bias_O

dense_2-kernelO

dense_2-bias_O-histogram

dense_2-kernelO-histogram

dense_bias_O-histogram

dense_kernelO-histogram

EXPERIMENTS **+**

Graph

Summary

Groupby

Horizontal Axis

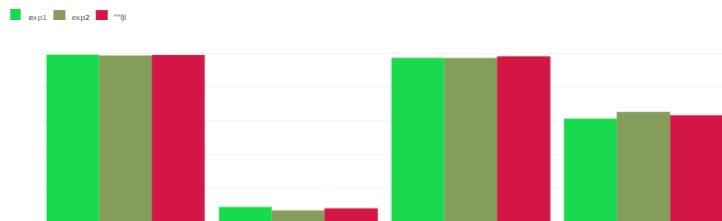
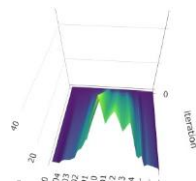
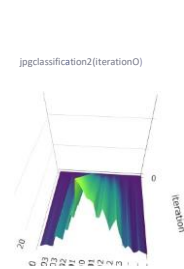
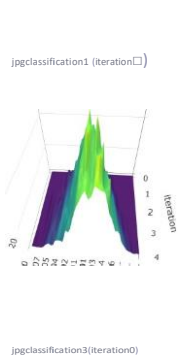
Iterations

Smoothing **e**

Exponential Moving Ave... **•**

Q

Summary



Side-by-side Debug Sample Comparison

Allows you to compare debug samples at different iterations to see how experiments perform as they run.

You can view debug samples by metric in the iterations shown, filtering the samples by selecting a metric from the drop-down menu. The most recent iteration appears first.

By clicking "Sync selection", you can synchronize the iteration and metric selection between experiments, so that selecting a metric from one experiment will automatically select the same metric for the other experiments in the comparison.

DETAILS HYPERPARAMETERS SCALARS PLOTS **DEBUG SAMPLES**

EXPERIMENTS +

Hyperparameter optimization
jpg classification 1 **fa89f4dc...**
opt. 2d8ecc16fb348c3a9c2982273453fa3 optimization
✓ Completed - 37,500 iterations - Last updated 2 years ago

Metric: testing Iterations: 12500 ↔ 37500

37500

1000-0_GT_frog_pred...	1000-1_GT_bird_pred_f...	1000-2_GT_bird_pred...
1000-3_GT_truck_pred...	1500-0_GT_deer_pred...	1500-1_GT_plane_pred...
1500-2_GT_horse_pred...	1500-3_GT_bird_pred...	2000-0_GT_horse_pred...

Hyperparameter optimization
jpg classification 2 **8c49c458...**
opt. 2d8ecc16fb348c3a9c2982273453fa3 optimization
✓ Completed - 37,500 iterations - Last updated 2 years ago

Metric: testing Iterations: 12500 ↔ 37500

37500

1000-0_GT_frog_pred...	1000-1_GT_bird_pred_f...	1000-2_GT_bird_pred...
1000-3_GT_truck_pred...	1500-0_GT_deer_pred...	1500-1_GT_plane_pred...
1500-2_GT_horse_pred...	1500-3_GT_bird_pred...	2000-0_GT_horse_pred...

Hyperparameter optimization
jpg classification 3 **36362ae8...**
opt. 2d8ecc16fb348c3a9c2982273453fa3 optimization
✓ Completed - 37,500 iterations - Last updated 2 years ago


Metric: testing Iterations: 12500 ↔ 37500

37500

1000-0_GT_frog_pred...	1000-1_GT_bird_pred_f...	1000-2_GT_bird_pred...
1000-3_GT_truck_pred...	1500-0_GT_deer_pred...	1500-1_GT_plane_pred...
1500-2_GT_horse_pred...	1500-3_GT_bird_pred...	2000-0_GT_horse_pred...

images - picasso

Iteration 100



w: 650 h: 650 x: y: Zoom: 116%

3.5 – Examples of Python for testing JupiterHUB -> ClearML execution

3.5.1 – example 1

```
from clearml import Task, Logger
import time

# Inizializza il Task ClearML
task = Task.init(project_name="GPU Project", task_name="TensorFlow GPU Training")
task.set_base_docker("tensorflow/tensorflow:2.17.0-gpu")
task.add_requirements("numpy", "1.26.4")
task.execute_remotely(queue_name="default")

#!pip install tensorflow[and-cuda]==2.17.0
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import socket
print(tf.__version__)

# Assicura l'utilizzo della GPU
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    print(f"GPU trovata: {gpus}")
    tf.config.experimental.set_memory_growth(gpus[0], True)
else:
    print(" ✕ Nessuna GPU trovata. Verifica l'installazione di TensorFlow.")

# Hyperparametri
params = {
    "epochs": 10,
    "batch_size": 64,
    "learning_rate": 0.001
}
task.connect(params)

# Dati di esempio (MNIST)
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Modello semplice
model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

model.compile(optimizer=keras.optimizers.Adam(learning_rate=params["learning_rate"]),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Addestra il modello monitorando la GPU
for epoch in range(params["epochs"]):
    history = model.fit(x_train, y_train, batch_size=params["batch_size"], epochs=1,
```



```
validation_split=0.2, verbose=2)
```

```
# Log delle metriche su ClearML
Logger.current_logger().report_scalar("Loss", "train", iteration=epoch,
value=history.history['loss'][0])
Logger.current_logger().report_scalar("Accuracy", "train", iteration=epoch,
value=history.history['accuracy'][0])
Logger.current_logger().report_scalar("Loss", "val", iteration=epoch,
value=history.history['val_loss'][0])
Logger.current_logger().report_scalar("Accuracy", "val", iteration=epoch,
value=history.history['val_accuracy'][0])

# Salva e carica il modello su ClearML
model.save("mnist_model.h5")
task.upload_artifact("MNIST Model", artifact_object="mnist_model.h5")

print("✅ Addestramento completato!")
task.close()
```

3.5.2 – example 2

```
from clearml import Task, Logger
import os
```

```
task = Task.init(project_name="GPU-Stress-test", task_name="GPU_Benchmark_01")
# Usare un'immagine Docker con driver cuda compatibili con quelli installati sul ClearML
Agent
task.set_base_docker('nvidia/cuda:12.3.0-base-ubuntu22.04')
task.execute_remotely(queue_name="default")
```

```
!pip install torch
import torch
import time
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Esecuzione sul device:", device)
```

```
size = (8192, 8192)
```

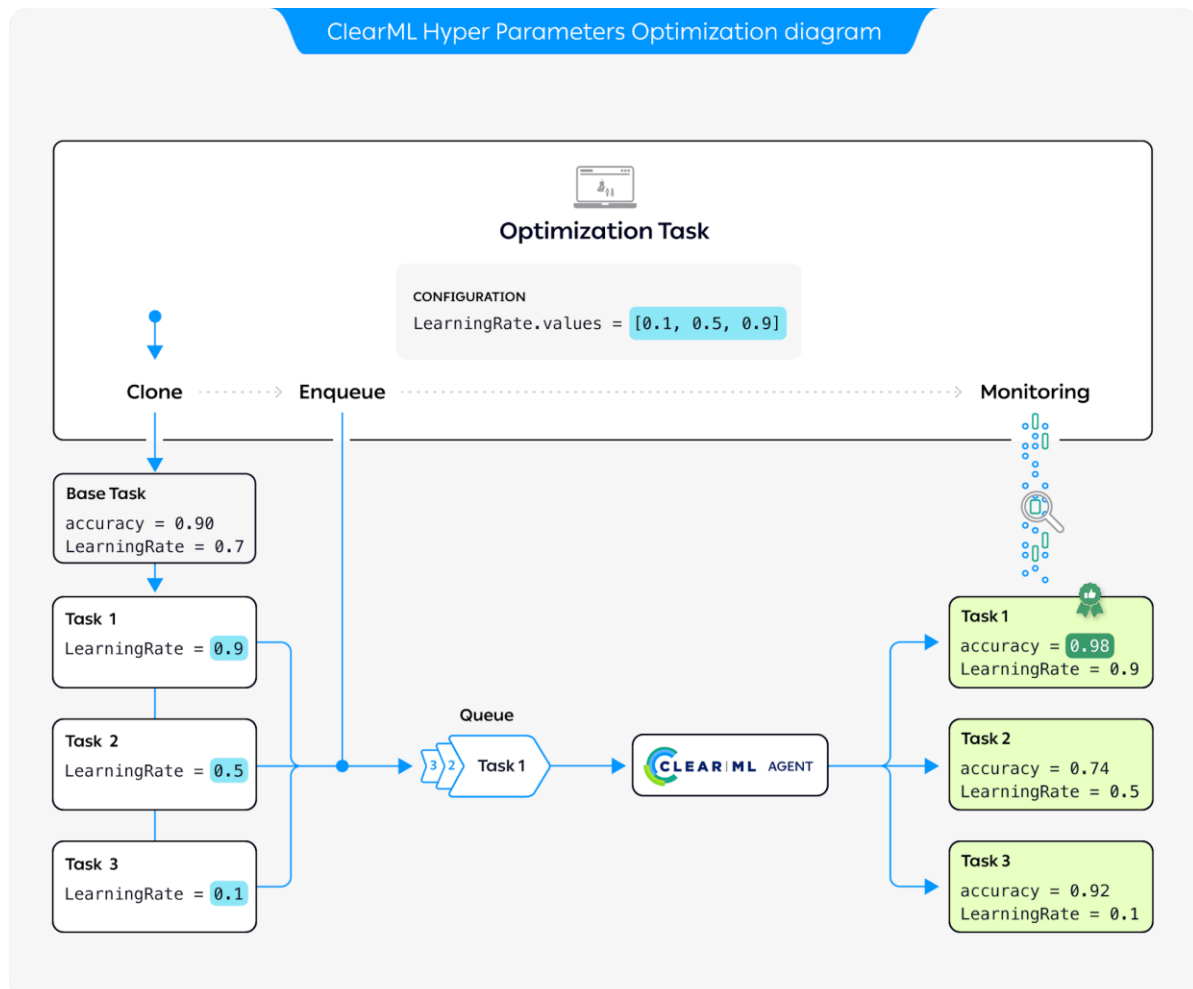
```
start_time = time.time()
ops = 0
print("start", start_time)
while time.time() - start_time < 60*5:
    tensor_0 = torch.randn(size, device=device, dtype=torch.float16)
    tensor_1 = torch.randn(size, device=device, dtype=torch.float16)
```

```
    result_0 = torch.matmul(tensor_0, tensor_0)
    result_1 = torch.matmul(tensor_1, tensor_1)
```

```
    torch.cuda.synchronize()
    ops = ops + 1
print("ops done:", ops, "in", (time.time() - start_time), "s")
task.close()
```

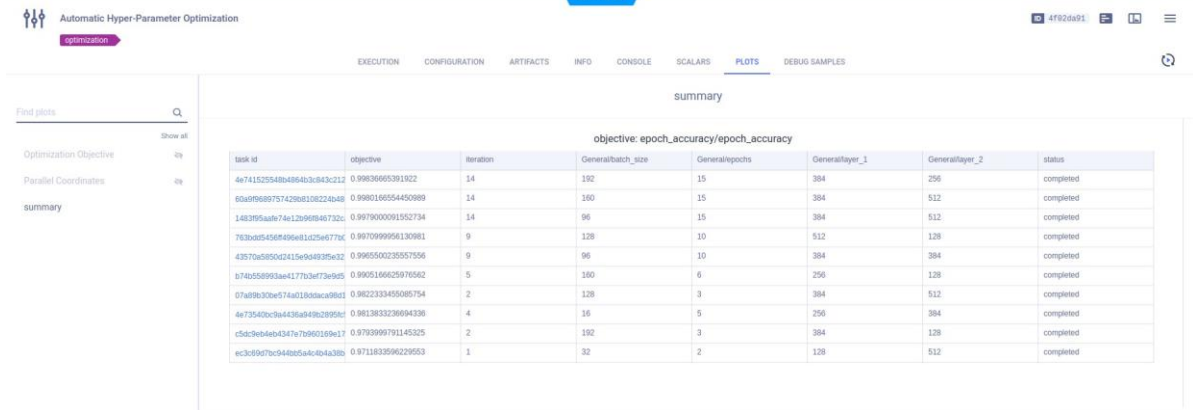
4. Hyperparameter Optimization

ClearML provides built-in tools for hyperparameter tuning, allowing developers to improve the performance of machine learning models through automated search techniques for the best configurations. Hyperparameter tuning is a fundamental step in the machine learning development cycle, as it allows you to systematically explore the parameter space, improving the model performance based on predefined metrics.



The diagram shows the typical flow of hyperparameter optimization, where parameters from a base Task are optimized:

1. Configure an Optimization Task with a base Task whose parameters will be optimized, and a set of parameter values to test.
2. Clone the base Task. Each clone's parameters are overwritten with a value from the Optimization Task.
3. Queue each clone for execution by a ClearML Agent.
4. The Optimization Task records and monitors the configuration and execution details of the cloned Tasks, and returns a summary of the optimization results in tabular and parallel coordinate formats, and in a scalar graph.



4.1 - Supported Optimizers

ClearML's HyperParameterOptimizer class includes modules for hyperparameter optimization. Its modular design allows the use of multiple optimizers, including existing frameworks, for simple, accurate, and fast hyperparameter optimization. Available optimizers include:

- **Optuna: is the default optimizer in ClearML, using various samplers such as grid search, random, Bayesian, and evolutionary algorithms.**
- **BOHB** : Combines the speed of Hyperband searches with the orientation and convergence guarantees of Bayesian optimization.
- **Uniform random sampling of hyperparameters** (RandomSearch in automation).
- **Full grid sampling strategy** of any combination of hyperparameters (GridSearch in automation).
- **Custom** : Use a custom class by inheriting from the ClearML automation base strategy class.

Supported Optimization Techniques

- **Grid Search:** Systematically explores all possible user-defined combinations of hyperparameters. Although it is an exhaustive method, Grid Search becomes computationally expensive when the hyperparameter space is very large.
- **Random Search:** This approach randomly extracts combinations of hyperparameters within a defined range. Although less exhaustive than Grid Search, Random Search is more efficient when the hyperparameter space is large, as it explores configurations in a less structured way but is more likely to find good results with fewer experiments.
- **Bayesian Optimization (Optuna):** This technique is based on probabilistic models that predict the impact of hyperparameters on model performance, guiding exploration towards the most promising configurations. This approach dramatically reduces the number of experiments needed to find the optimal parameters compared to Grid or Random Search.
- **BOHB:** This technique combines Bayesian optimization with a resource selection approach based on HyperBand. This means that BOHB not only finds good hyperparameter configurations, but also optimizes the use of computational resources, making it particularly suitable for resource-constrained environments.

ClearML Integration

ClearML simplifies the use of optimizers within workflows by allowing users to define the parameters to optimize directly in the GUI or via the Python SDK. Once the objective function and the hyperparameter range are defined, ClearML automatically creates the necessary tasks and starts the optimization.

4.1.1 - Comparison task

The basic idea is to use a dedicated task that retrieves the hyperparameters and metrics of experiments already done to compare them programmatically.

This method would allow us to produce any table and graph (as long as it is possible to do so with Python).

By exploiting the cloning and reproducibility of experiments features provided by ClearML, if adequately parameterized, it would be possible to run the comparison task multiple times without having to intervene on the code.

Accessing Tasks

A task can be identified by its project and name, as well as a unique identifier (UUID string). A task's name and project can be changed after an experiment is run, but its ID remains the same.

Programmatically, task objects can be retrieved by querying the system based on the task ID or a combination of project and name, using the `Task.get_task` class method. If a combination of project/name is used and there are multiple tasks with the same name, the function will return the last modified task.

Examples:

- Using ID:

```
a_task = Task.get_task(task_id='123456deadbeef')
```

- Using name and project:

```
a_task = Task.get_task(project_name='examples', task_name='artifacts')
```

Once you have a task object in ClearML, you can query the task's state, reported scalar values, etc. You can also retrieve the task's outputs, such as artifacts and models. This provides an efficient way to analyze and manage the data and results generated by each task.

Search and filter tasks programmatically

Enter the search parameters into the `Task.get_tasks` method, which returns a list of task objects that match the search.

```
task_list = Task.get_tasks(  
    task_ids=None, # type Optional[Sequence[str]]  
    project_name=None, # Optional[str]  
    task_name=None, # Optional[str]  
    allow_archived=True, # [bool]  
    task_filter=None, # Optional[Dict]#  
    # tasks with tag `included_tag` or without tag `excluded_tag`  
    tags=['included_tag', '-excluded_tag']  
)
```

You can filter tasks by passing rules to `task_filter`

```
task_filter={  
    # filter out archived tasks  
    'system_tags': ['-archived'],  
    # only completed & published tasks  
    'status': ['completed', 'published'],  
    # only training type tasks  
    'type': ['training'],  
    # match text in task comment or task name  
    'search_text': 'reg_exp_text',  
    # order return task lists by their update time in ascending order  
    'order_by': ['last_update']  
}
```

Using Artifacts

A task's artifacts are accessible via the task's artifact property, which lists the artifact locations. Artifacts can then be retrieved from their locations using:

- `get_local_copy()`: Downloads the artifact and caches it for later use, returning the path to the cached copy.
- `get()`: Returns a Python object constructed from the downloaded artifact file.

```
# get instance of task that created artifact, using task ID
preprocess_task = Task.get_task(task_id='the_preprocessing_task_id')
# access artifact
local_csv = preprocess_task.artifacts['data'].get_local_copy()
```

Accessing Parameters

To access all parameters of a task, use the `Task.get_parameters` method. This method returns a flattened dictionary of 'section/parameter': 'value' pairs.

```
task = Task.get_task(project_name='examples', task_name='parameters')

# will print a flattened dictionary of the 'section/parameter': 'value'
# pairs
print(task.get_parameters())
```

To access a specific parameter in ClearML, use the `Task.get_parameter` method specifying the parameter name and section.

```
param = task.get_parameter(name="Args/batch_size")
```

Retrieving Scalar Values

Scalar Summary

Use the `Task.get_last_scalar_metrics()` method to get a summary of all scalars logged in the task. This call returns a nested dictionary of the last values, as well as the maximum and minimum values reported for each scalar metric logged in the task, sorted by title and series. This provides a detailed and up-to-date overview of the task's performance in terms of different scalar metrics.

Get Sample Values

To get a sample of the scalar values logged in a task, use the `get_reported_scalars()` method. This method allows you to retrieve a sample of the scalars logged for each metric/series.

You can specify the maximum number of samples per series to return (up to 5000) using the `max_samples` argument.

To retrieve all scalar values, use `Task.get_all_reported_scalars()` instead. Additionally, you can set the x-axis units with the `x_axis` argument, choosing between iteration (default), timestamp (milliseconds since epoch), or `iso_time` (real time).

Get single-valued scalars

To get the values of single-valued scalars logged in a task, use the `Task.get_reported_single_value()` method, specifying the scalar name.

If you want to get all logged single-valued scalars, use `Task.get_reported_single_values()`, which returns a dictionary of scalar name and value pairs. This method is useful for analyzing specific scalar results within a task.

5. ClearML Feature Testing

After selecting ClearML as the primary MLOps tool for Snap4City, we ran a series of practical tests to evaluate its effectiveness in dataset versioning and experiment comparison capabilities, two critical aspects to ensure reproducibility and detailed analysis of machine learning models.

5.1 - Dataset Versioning

ClearML offers advanced dataset versioning capabilities, allowing you to manage and track different versions of the datasets used for experiments. During the tests, we used an image classification dataset to train a computer vision model, leveraging ClearML Data's capabilities to version the datasets and ensure their accessibility from any machine.

One of the main advantages of this system is that each experiment can draw on a specific version of the dataset, ensuring full reproducibility. If the dataset is modified, ClearML Data only records the differences from the previous version, thus optimizing storage space and maintaining full traceability of the changes made to the dataset. Figure 2 22 and Figure 2 23 show the dataset management interface for this specific test.

We found that once a dataset is finalized in ClearML, it becomes immutable, providing an additional level of data security and integrity. This allows future experiments to use the exact data from previous versions, improving collaboration and reuse of work, as well as ensuring that results are always repeatable and verifiable.

This advanced dataset management capability has proven particularly useful for complex computer vision experiments, where ensuring that models and data are tightly aligned and accurately versioned is essential.

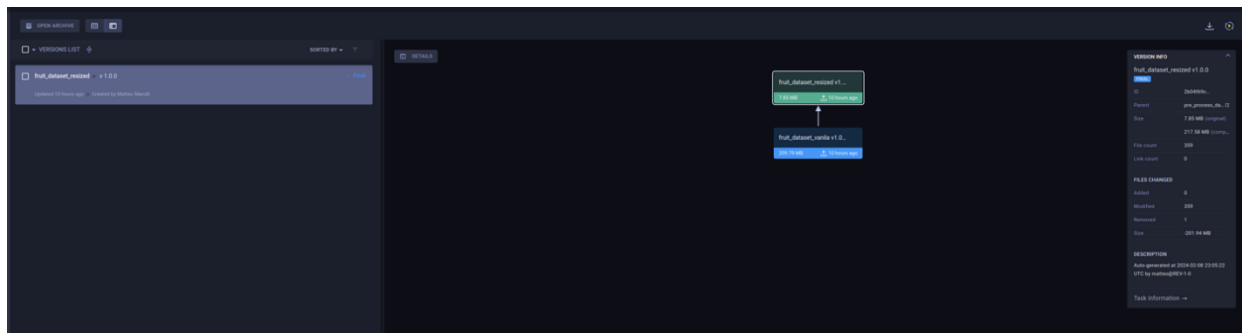


Figure 2 22 Dataset Management Test – Versioning Interface

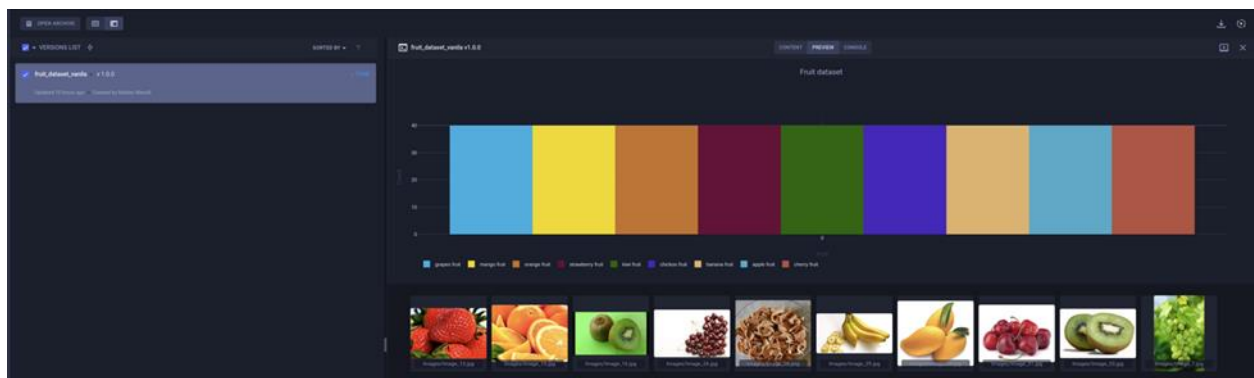


Figure 2 23 Image Dataset Management Test – Dataset Preview Interface

5.2 - Comparing Experiments

Comparing experiments is one of the core capabilities of ClearML, allowing for in-depth and intuitive analysis of model results and parameters. During testing, we used ClearML's Experiments Table to

compare hyperparameters, metrics, and performance across different model iterations in real time. This was especially useful for tracking differences between experiments and assessing the impact of changes on machine learning algorithms.

ClearML allows you to visualize experiments in a leaderboard view, with the ability to customize columns, filter results, and sort models by specific criteria such as accuracy, loss, or execution time.

Another key feature we tested was the ability to clone experiments: once you've identified an experiment that performs well, you can clone it, change its parameters or dataset, and rerun it. This allowed you to quickly iterate on model configurations while providing full traceability of previous versions and new experiments derived from them.

ClearML also supports scalar plots and parallel coordinates, which were useful tools for interactively visualizing the impact of hyperparameters on results. These plots allow us to overlay the results of multiple experiments and visually analyze how small changes in hyperparameters affect the final performance. The ability to visually compare experiments accelerated the model tuning process, helping us quickly identify the best configurations.

During these tests, the idea of using a comparison task also emerged. Through this dedicated task, the comparison of multiple experiments could be automated, collecting and aggregating key metrics and visualizing the results in a standardized format. Implementing a comparison task would allow for periodic evaluations of the most recent experiments, making it easier to review and choose the best configurations based on specific criteria. This task could be integrated into existing workflows, further streamlining the model comparison and selection process.

6. Inference API

The two main APIs developed for ClearML Serving were specifically designed to meet the needs of Snap4City, providing flexible solutions for queuing tasks and inference requests.

6.1 On-Demand API

The **On-Demand Service** is designed to respond to inference requests in real time, exploiting a DAP ready to respond loaded in memory on some server. The requested model is pre-loaded in memory (usually in the GPU) and ready to perform inference without the need to wait for resources to load.

Flow of the On-Demand API:

1. The API receives an http request with the hash of the task handling the inference on a specific machine (to locate a specific machine), the name of the endpoint to be called, the input of the endpoint and the access_token of Snap4City.
2. Validation of the access token via Snap4City is performed to ensure that only authorised users can make a request.
3. Using the hash provided, the IP address of the specific machine is traced back to the ClearML Utils database.
4. A request is made to the IP obtained with the name of the endpoint provided.
5. The result of the request is returned to the user.

In this fragment, a POST request is made to the pre-loaded model endpoint, and the result is returned in real time:

```
machine_id = body.get("machine_id", "")
if not machine_id:
    self.log_result("", "", False, {"error": "Machine id is missing"})
    return 'Error: Machine id is missing'
# Extract the endpoint from the body
hashed_endpoint = body.get("endpoint", "")
if not hashed_endpoint:
    self.log_result("", "", False, {"error": "Endpoint is missing"})
    return 'Error: Endpoint is missing'
log_details = {"params": body.get("params", {})}
status = False
endpoint_task_id = ""
try:
    endpoint_task_id = self.decode_endpoint(machine_id, hashed_endpoint)
    base_url = self.retrieve_base_url_mapping(endpoint_task_id)
except ValueError as e:
    log_details["error"] = str(e)
    self.log_result("", hashed_endpoint, status, log_details)
    return f "Error: Invalid Endpoint ID or Endpoint Task ID: {hashed_endpoint}"
if not base_url:
    log_details["error"] = "No base_url mapping found".
    self.log_result(endpoint_task_id, hashed_endpoint, status, log_details)
    return f "Error: No base_url mapping found for Endpoint Task ID: {endpoint_task_id}"
url = f"{base_url}/{hashed_endpoint}"
# Make a POST request to the decoded URL with the params from the body
try:
    response = requests.post(url, json=body.get("params", {}))
    response.raise_for_status() # Raise an exception for HTTP errors
    response_data = response.json()
    log_details.update(response_data)
    status = True
except requests.RequestException as e:
    log_details["error"] = str(e)
    self.log_result(endpoint_task_id, hashed_endpoint, status, log_details)
    return f "Error: Failed to make POST request to {url}. Error: {str(e)}"
```

6.2 API Task Enqueue/Sporadic

The **Enqueue/Sporadic Task Service** allows users to send a task to the execution queue, delegating execution to ClearML agents. The API receives **the task's hash ID**, the queue name and any input for the task. Once the request is received, the task is cloned, configured with any parameters and placed in the selected queue, ready to be executed by the ClearML agent as container deployed on the fly.

API flow:

1. The API receives a request containing: the hash ID of the task, the queue name, the input for the task and the Snap4City access_token.
2. Access token validation via Snap4City is performed to ensure that only authorised users can submit tasks.
3. The hash is translated with the real id of the task in the ClearML Utils database.
4. The task is cloned and queued.
5. The API returns a response with the status of the operation.

Code fragment showing how the API clones the original task, connects it to the given parameters and places it in the specified queue:

```
hashed_task_id = body.get("task_id", "")
if not hashed_task_id:
    return 'Error: Task ID is missing'.
try:
    task_id, default_queue = self.decode_task_id(hashed_task_id)
    task = Task.get_task(task_id=task_id)
except ValueError as e:
    return f "Error: Invalid Task ID: {hashed_task_id}"
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
cloned_task = Task.clone(source_task=task, name=f"{task.name}_{timestamp}")
cloned_task.connect(body.get("params", {}))
cloned_task.set_base_docker(None)
queue = body.get("queue") if body.get("queue") else default_queue
Enqueue/Sporadic_response = Task.Enqueue/Sporadic(task=cloned_task, queue_name=queue)
```

6.3 ClearML Utils

The webapp, available at <http://192.168.1.xxx:9000>, is designed for internal use. This application allows administrators and developers to easily manage ClearML resources configured for serving and task Enqueue/Sporadic, as well as providing an interface to monitor service calls and view detailed logs.

6.3.1 - Technology Used: Stack TALL and Filament

The **TALL** stack combines modern technologies that simplify the development of dynamic and interactive web applications:

- **Tailwind CSS:** A highly customisable CSS framework that facilitates the creation of responsive, modern user interfaces with a utility-first approach. It allows you to quickly create elegant layouts without writing complex CSS.
- **Alpine.js:** A lightweight JavaScript framework that provides responsive functionality without the need for a heavy front-end framework such as Vue.js or React. It is perfect for handling dynamic client-side interactions with a simple and readable syntax.
- **Laravel:** The popular PHP framework for backend development, known for its simplicity, elegance and wide range of built-in functionality. Thanks to its MVC (Model-View-Controller) architecture, Laravel facilitates the structured and modular development of web applications (Laravel Docs, 2024).

- **Livewire:** A key component of the stack, allowing you to create responsive user interfaces without having to write complex JavaScript. Livewire allows interfaces to be updated in real time by communicating with the Laravel backend, greatly simplifying server-side logic management and application interactivity.

In addition to these tools, the webapp uses **Filament**, a Laravel extension designed to build administrative panels and CRUD (Create, Read, Update, Delete) applications quickly and scalably.

6.3.2 - Advantages of Filament

Filament is a toolkit that integrates seamlessly with Laravel and offers a number of significant advantages for the development of in-house management applications, such as the one developed for ClearML resource management:

- **Speed of development:** The package includes code generators and predefined components that accelerate the development process. In just a few steps, ClearML API and resource management interfaces could be created.
- **Extendability and maintenance:** Filament integrates seamlessly with Laravel and Livewire, ensuring that any new functionality can be added in a modular manner without breaking the existing architecture. This keeps the code clean and easy to extend in the future.

6.4 Main ClearML Dashboard

The **dashboard** provides an overview of available resources and the use of ClearML services. The main features of the dashboard include:

- **Number of configured serving machines:** Displays the total number of machines configured for on-demand serving.
- **Number of endpoints for on-demand service:** Shows the number of endpoints configured and ready for immediate inference.
- **Number of tasks available for the Enqueue/Sporadic task:** Lists the tasks configured for use by end users via the task Enqueue/Sporadic API.
- **Graph of requests to the task Enqueue/Sporadic service:** Displays a graph tracking the number and trend of requests made to the task Enqueue/Sporadic service.
- **Graph of requests to the on-demand service:** Shows the trend of requests made to the on-demand inference service.

6.5 ClearML Serving Machine Management

An interface (Figure 6-1, Figure 6-2) allows administrators to manage the machines configured for serving. On this page, administrators can:

- **Enter the IP address of the machine** on which ClearML Serving is configured.
- **Enter the serving task ID:** This information is used to map the IP of the machine to the serving task, allowing the application to correctly call the served models.

This feature simplifies resource management, allowing new machines to be easily registered and associated with configured serving tasks.

ClearML Utils

A

Figure 6-1 ClearML Utils - list of machines configured for serving.

ClearML Utils

A

Figure 6-2 ClearML Utils - Screen shot of adding a serving machine.

6.6 On-Demand Endpoint Management

Another key functionality of the webapp is endpoint management for the on-demand service (Figure 6-3, Figure 6-4, Figure 6-5). Developers, once an endpoint has been generated via **ClearML Serving**, can:

- **Enter the name of the endpoint** and the ID of the associated serving task.
- The system generates a **hash of the serving task**, which will be used by end users to interact with the endpoint without the need to know the actual task ID.

This solution improves security and ease of use, allowing the end user to use the endpoint via a hash without having to know the technical details of the task.

ClearML Utils

A

Endpoint Mappings > Lista

Endpoint Mappings

<input type="checkbox"/> Serving machine task id	Endpoint	Hashed machine id	Description
<input type="checkbox"/> 2655bb0a64fa4c57be93637bd6441b9e	miaendpoint	fcff29a163839aa8ce74f7cdb210bc...	Modifica

Mostrati da 1 a 1 di 1 risultati

per pagina 10

Figure 6-3 ClearML Utils - List of endpoints

ClearML Utils

A

Endpoint Mappings > Nuovo

Nuovo Endpoint Mapping

Serving Machine Task ID*

Nome Endpoint*

Descrizione

Salva Salva & nuovo Annulla

Figure 6-4 ClearML Utils - Adding new endpoint

ClearML Utils

A

Endpoint Mappings > Modifica

Modifica Endpoint Mapping

Serving Machine Task ID

Nome Endpoint

Hashed Machine ID Id pubblico da utilizzare per identificare la macchina servente

Descrizione

Salva Annulla

Elimina

Figure 6-5 ClearML Utils - Display hash endpoints

6.7 Task Management for the Task Enqueue/Sporadic Service

The webapp also offers an interface (Figure 6-6, Figure 6-7, Figure 6-8) to manage the tasks available for the task Enqueue/Sporadic service. Developers, in order to offer the possibility of cloning and queuing a task, must:

- Access the interface and enter the **task ID** and **default queue name**.
- The application generates a **hash of the task ID**, which the end user will use to call the task Enqueue/Sporadic service.

This system allows end users to start cloned tasks using only the hash, without having to access the original task and know the full technical details.

ClearML Utils

A

Figure 6-6 ClearML Utils - List of tasks configured for use via API Task Enqueue/Sporadic

ClearML Utils

A

Figure 6-7 ClearML Utils - Screen shot for adding a new task

ClearML Utils

A

Figure 6-8 ClearML Utils - Screen to display the details of the configured task and retrieve the hash created by the application

6.8 Service Call Logs

The webapp provides two sections dedicated to displaying service call logs, allowing administrators and developers to monitor traffic and the status of requests:

- **Log of calls to the on-demand service** (Figure 6-9Figure 6-10): In this section, you can view all requests made to the on-demand service, including information such as:
 - Success or failure of the request.
 - Date and time of the request.
 - Serving machine and endpoint called.
 - Response received from the service.
- **Log of calls to the task Enqueue/Sporadic service** (Figure 6-11Figure 6-12): Shows all requests made to the task Enqueue/Sporadic service, including information such as:

- ID of the cloned task.
- Success or failure of the request.
- Details of the response received by the system.

These sections allow greater transparency and control over the use of services, facilitating debugging and optimisation of operations.

ClearML Utils

Dashboard > Ondemand Logs > Lista

Ondemand Logs

<input type="checkbox"/> Serving machine task id	Endpoint	Status	Created at	
<input type="checkbox"/> b746714516a7409bbaa68a8402a5c1a2	nome_endpoint	✓	10/07/24 12:31:45	👁️ Vedi

Mostrati da 1 a 1 di 1 risultati

per pagina 10

1 (points to Ondemand Logs in sidebar)

2 (points to Vedi icon)

Figure 6-9 ClearML Utils - List of OnDemand/Stable service call logs.

ClearML Utils

Dashboard > Ondemand Logs > Guarda

Guarda Ondemand Log

Data Creazione: 10/07/2024, 12:31:45

Serving Machine Task ID: b746714516a7409bbaa68a8402a5c1a2

Nome Endpoint: nome_endpoint

Successo: ☒ Yes ☐ No

Dettaglio:

Figure 6-10 ClearML Utils - Detail of an OnDemand/Stable service log.

ClearML Utils

Dashboard > Enqueue Task Logs > Lista

Enqueue Task Logs

<input type="checkbox"/> Task id	Status	Created at	
<input type="checkbox"/> 2655bb0a64fa4c57be93637bd6441b9e	✓	10/07/24 10:20:02	👁️ Vedi

Mostrati da 1 a 1 di 1 risultati

per pagina 10

1 (points to Enqueue Task Logs in sidebar)

2 (points to Vedi icon)

Figure 6-11 ClearML Utils - List of call logs to the Task Enqueue/Sporadic service.

Dashboard

Task Mappings

Endpoint Mappings

Serving Machine Mappings

Enqueue Task Logs

Ondemand Logs

Enqueue Task Logs > Guarda

Guarda Enqueue Task Log

Data Creazione

10/07/2024, 10:20:02

Task ID

2655bb0a64fa4c57be93637bd6441b9e

Successo

✓ Yes

✗ No

Dettaglio

Figure 6-12 ClearML Utils - Detail of a Task Enqueue/Sporadic service log.

7. Use of Services AI/DA from IoT App/Proc.Logic

These blocks require users to obtain an access-token from Snap4City, which is then validated by the ClearML API. This solution ensured that only authorised users can access and interact with the services, while maintaining a high level of security within the Snap4City platform.

The adoption of the authenticated version of the blocks allowed ClearML services to be securely integrated into existing workflows on Snap4City, making the solution suitable for use in both experimental and operational environments.

The two Node-RED blocks developed are:

1. **Enqueue/Sporadic Task Block (Figure 7-1Figure 7-2):** This block allows a task to be sent to the specified queue, using an input JSON that includes the task_id, queue_name and the necessary input parameters for the task. The output of the block is a JSON indicating the status of the task (ok or ko) and the response of the task.Enqueue/Sporadic function.

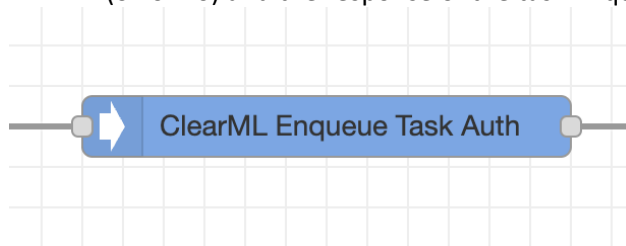


Figure 7-1 Node-RED block for the Enqueue/Sporadic Task service.

Authentication

task_id

queue_name

Input

Figure 7-2 Node-RED Enqueue/Sporadic Task Block - Detail of input parameters.

Example of input:

```
{
  "task_id": "hash_del_task_id",
  "queue_name": "queue_name",
  "input": {
    "parameter_1": "value",
    "parameter_2": "value".
  }
}
```

Code fragment that makes the request to the API to queue the task:

```
var body = {
  access_token: accessToken,
  task_id: task_id,
  queue: queue_name,
  params: input_params
};
```

```
// Make the HTTP POST request
```



```
request.post({
  url: 'https://www.snap4city.org/clearml/serve/Enqueue/Sporadic_task',
  headers: {
    "accept": "application/json",
    "Content-Type": "application/json"
  },
  body: JSON.stringify(body) // Convert body object to JSON string
}, function(error, response, body) {
  if (error) {
    const errorMsg = "HTTP request error: " + error.message;
    logger.error(errorMsg, { error: error, msg: msg });
    node.error(errorMsg, msg);
    msg.payload = {
      status: "ko",
      error: error.message
    };
    node.send(msg); return;
  }
}
```

2. **Block for On-Demand Service** (Figure 7-3Figure 7-4): This block calls an on-demand service for immediate inference, using a JSON that contains the hash ID of the serving machine (`machine_id`), the model endpoint (`endpoint`) and the input parameters for inference. The output of the block is a JSON with the status and response of the service.

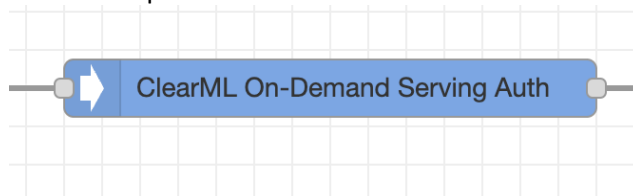


Figure 7-3 Node-RED block for On-Demand service.

Authentication

machine_id

endpoint

input

```
{}
```

Figure 7-4 Node-RED On-Demand block - Detail of input parameters.

Example of input:

```
{
  "machine_id": "hash_del_serving_machine_task_id",
  "endpoint": "endpoint_name",
  "input": {
    "parameter_1": "value",
    "parameter_2": "value".
  }
}
```

Code fragment making the call to the On-Demand API

```
var body = {
```

```
    access_token: accessToken,
    machine_id: machine_id,
    endpoint: endpoint,
    params: msg.payload.input || config.input
  };

request.post({
  url: 'https://www.snap4city.org/clearml/serve/OnDemand/Stable',
  body: body,
  json: true // Ensures the body is sent as JSON and the response is parsed as JSON
}, function (error, response, body) {
  if (error) {
    const errorMsg = 'Error in ClearML On-Demand Serving Auth node: ' +
error.message;
    logger.error(errorMsg, { error: error, msg: msg });
    node.error(errorMsg, msg); msg.payload = {
      status: 'ko',
      error: error.message
    };
  } else {
```

7.1 Authentication with Snap4City

To ensure that only authorised users can access ClearML Serving services, robust authentication via Snap4City was implemented. Before Node-RED blocks can be used to queue tasks or request on-demand inferences, users must authenticate with Snap4City.

The authentication process takes place via an **access-token** provided by Snap4City. When a user interacts with Node-RED blocks, the block automatically requests an access-token from Snap4City, which is then transmitted to the ClearML API at the time of the request. On the server side, the API validates the access-token to verify that the user is authorised to access the requested services.

This configuration made it possible to offer flexible and scalable inference services, both through queued and on-demand tasks, while maintaining a high level of security. Integration with Snap4City and centralised management via Node-RED made these services easily accessible and seamlessly integrated with the ClearML ecosystem.

7.2 Remote task notification system

The second tool developed is a task running in service mode that constantly monitors the status of tasks executed on ClearML agents. This task sends automatic notifications via **Skype** to administrators when a task:

- It is successfully completed.
- It fails during execution.

Each notification includes the last lines of the task's console log, providing an immediate and useful account of what happened during execution.

Operation of the Monitoring Task

1. **Constant monitoring:** The task remains running as a service and monitors the tasks launched on ClearML agents. Using the ClearML API, the task monitors the status of each running task.
2. **Sending notifications:** When a task is completed or fails, the system generates a notification on Skype. This allows administrators to receive real-time updates on task status without having to manually access the ClearML interface to monitor tasks.
3. **Integration with task logs:** In addition to the general status of the task, the notification includes the last lines of the console log, giving an immediate idea of the reason for the success or failure of the task.

7.2.1 - Executing the Task in Service Mode

ClearML agents support **Service** mode, a configuration that allows multiple tasks to be executed at the same time, as opposed to standard mode where an agent executes only one task at a time. In standard mode, agents are generally used for training models, processes that require the intensive use of all available resources. In contrast, **Service** mode is designed to perform light tasks that consume few resources and remain idle most of the time. Examples of such tasks include periodic cleaning services, notification services or pipeline controllers, which monitor or manage the workflow without significantly impacting system resources. This mode is particularly useful for ensuring the continuous operation of support and monitoring services, while maintaining the availability of resources for more intensive tasks. An example of a command for starting an agent in Service mode:

```
clearml-agent daemon --services-mode --queue services --create-queue --docker ubuntu:22.04 --cpu-only -detached
```

7.2.2 - Integration with Skype

The integration with Skype for sending automated notifications was realised using the Python library **SkPy** (SkPy Docs, 2024). This library provides a simple API to interact with Skype, allowing authentication, chat management and message sending directly from a Python application. SkPy supports both individual and group chats, making it ideal for real-time notifications about tasks performed in ClearML.

Importing the library: To begin with, we import the SkPy library with the necessary modules for accessing and managing Skype messages.

```
from skpy import Skype, SkypeMsg
```

Logging in to Skype: The Skype account is logged in using the credentials of a bot specifically created for the lab, which sends notifications. SkPy supports both username and password authentication and the use of tokens, but for simplicity's sake the username and password approach is used here.

```
self.sk = Skype('clearml-bot-disit@outlook.it', '***')
```

Selection of the correct chat: Once logged in, the specific group chat where notifications will be sent is selected. The chat is identified by its unique ID, passed as an argument to the script.

```
self.chat = self.sk.chats.chat(chat) # chat is the id of the group chat in which to send notifications.
```

Message formatting: The message to be sent is formatted using Skype's rich formatting, which allows links, bold text and monospace to be added for better readability. For example, the link to the experiment log, task status, project name and task name is provided, along with the console output.

```
message = "{}Experiment ID {} {}Project: {} - Name: {}".format\n( self._message_prefix, SkypeMsg.link(task.get_output_log_web_page(),task.id), SkypeMsg.bold(task.status),\nSkypeMsg.bold(task.get_project_name()), SkypeMsg.bold(task.name),\nSkypeMsg.mono("\n".join(console_output))[-2048:])
```

Sending the message: Once formatted, the message is sent to the selected group chat. The parameter `rich=True` allows the message to be sent with enriched formatting.

```
self.chat.sendMessage(message, rich=True)
```

7.2.3 - Starting the Task

The Python script is executed with the chat ID as argument. You can also include an option to receive notifications about successfully completed tasks, not just failed ones.

```
python skype_alerts.py --chat '19:1cb604afed954eb6a1e4c7ed9bdc0cfe@thread.skype' --\ninclude_completed_experiments
```

7.2.4 - Advantages of the SkPy Library:

- **Ease of use:** SkPy greatly simplifies integration with Skype, enabling authentication, chat management and messaging in just a few steps.
- **Rich Messaging:** SkPy supports advanced message formatting (links, bold text, monospace), making notifications more readable and informative.
- **Automation:** Thanks to SkPy, it is possible to fully automate the sending of notifications, integrating them into Python scripts that monitor ClearML tasks and inform users in real time.

This solution proved to be very effective in keeping lab users up-to-date on task status, allowing them to receive timely notifications of any errors or experiment completions directly in a Skype group chat.

```
Experiment ID b333b112dd544567b773ec06fb000f0d completed
Project: Cloned for Serving - Name: print_message_test_20240710_160628_20240711_140649

e/pypoetry
agent.docker_internal_mounts.vcs_cache = /root/.clearml/vcs-cache
agent.docker_internal_mounts.venv_build = ~/.clearml/venvs-builds
agent.docker_internal_mounts.pip_download = /root/.clearml/pip-download-cache
agent.apply_environment = true
agent.apply_files = true
agent.custom_build_script =
agent.disable_task_docker_override = false
agent.git_user =
agent.git_pass = ****
agent.default_python = 3.10
agent.cuda_version = 0
agent.cudnn_version = 0
```

Figure 7-5 Example Notification Message on Skype

8. Testing and Validation

This chapter provides a concrete example of the development and implementation cycle of a machine learning model, using ClearML to manage the training, deployment and monitoring process. The aim is to demonstrate how the system meets the requirements for access by third-party users, use on authorised machines and monitoring by administrators.

8.1 Model Development and Training

For this test, the example provided by ClearML **Train and Deploy Pytorch model with Nvidia Triton Engine** was used (ClearML Serving Example GitHub, 2024). The model used is a convolutional neural network (CNN) for digit recognition, using the **MNIST** dataset, one of the most popular datasets in the field of computer vision.

The development of the model took place on **JupyterHub**, where a ClearML task was initialised and sent the training code to agents configured on machines equipped with GPUs. The selected agent managed the training automatically, making maximum use of the available resources and recording each phase of the training on ClearML.

Training Steps:

- Virtual environment created on JupyterHub with the necessary packages installed.
- Task ClearML initialised for model training using the command:

```
python train_pytorch_mnist.py
```
- Once completed, the trained model was automatically saved to ClearML, making it available for subsequent inference tasks.

8.2 Deploying the Model on ClearML Serving

After training, the model was put into production using ClearML Serving. The deployment process involves two main steps:

Configuration of the Serving Machine

An administrator must configure a machine for serving using ClearML. This operation is only performed once per machine. The administrator configured the machine by executing the following commands:

```
docker-compose --env-file example.env -f docker-compose-triton-gpu.yml up
```

Subsequently, the machine was registered on **ClearML Utils** (see Chapter 0) by entering the IP address and ID of the machine's inference task to make it available to developers.

Creation of the Endpoint for Inference

The developer created an inference endpoint for the model trained with ClearML Serving via the following command:

```
clearml-serving --id <serving_service_id> model add --engine triton --endpoint "<model_name>" --preprocess "preprocess.py" --name "<model_name>" --project "<project>"
```

After the endpoint was created, it was registered on **ClearML Utils** by entering the serving task id and endpoint name (see Chapter 0) to allow third-party users to access it easily without needing to know the technical details of the task or the IP address of the machine.

8.3 Testing with Node-RED Block

To test the on-demand inference, the **Node-RED block** developed specifically to interact with the On-Demand API was used. This block allows the user to send images as input and receive the model prediction as output. The Figure 8-1 shows the flow realised to test the service.



Figure 8-1 Test - Node-RED flow with the On-Demand service block

- **Input** (Figure 8-2):
 - An image of a figure taken from the MNIST dataset.
 - Hash of the machine id
 - Endpoint name

```

{
  "machine_id": "05425cc1f3729b542b6538e34187dcec836890352edaa42b8531194e705ebb07",
  "endpoint": "test_mnist_model_pytorch_61",
  "input": {
    "url": "https://raw.githubusercontent.com/allegroai/clearml-serving/main/examples/pytorch/5.jpg"
  }
}
  
```

Figure 8-2 Test - Input of the request to the On-Demand service to make inference with a digit recognition model.

- **Output** (Figure 8-3):
 - The figure predicted by the
 - Request status

```

▼ object
  status: "ok"
  ▼ response: object
    digit: 5
  
```

Figure 8-3 Test - Response of the request to the On-Demand API with a pattern for digit recognition.

8.4 Testing On-Demand Inference API with CSBL

The on-demand inference API can be tested also by exploiting **CSBL**, as illustrated by the following Javascript code example. A POST request is made to call the ClearML on-demand Inference API on a LLM which has been previously deployed in production with ClearML Serving (see Section 8.2).

In order to make the POST request, first the access token of the user logged in Snap4City has to be retrieved. Then, a simple text prompt (e.g.: "What are recurrent neural networks?" can be wrapped in the POST body message, which should be of the following form:

```

var body = {
  access_token: <SNAP4CITY_ACCESS_TOKEN>,
  machine_id: "<HASHED_MACHINE_ID>",
  endpoint: "<ENDPOINT_NAME>",
  params: {"prompt": "What Are Recurrrent Neural Networks?"}
};
  
```

where "<HASHED_MACHINE_ID>" is the hashed machine id of the ClearML agent which is serving the on-demand Inference API and "<ENDPOINT_NAME>" is the name used to create the endpoint with the specific model (in this case, a pre-trained LLM), as described in Section 8.2. The structure of the data

passed in the "params" parameter of the body depends on how the deployed model accepts input data. This is typically defined and managed in the preprocess python file specified when the model endpoint is created (i.e. the "preprocess.py" file shown in Section 8.2).

The following snippet code shows how to implement all the previously described procedure. The response of the API is then simply showed in an alert message box. In order to work properly, a valid "<HASHED_MACHINE_ID>" and a valid "<ENDPOINT_NAME>" must be provided.

```
// Get Snap4City Access Token
$.ajax({
  url:
    "https://www.snap4city.org/dashboardSmartCity/controllers/getAccessToken.php",
  type: "GET",
  async: true,
  dataType: 'json',
  success: function(dataSso) {
    let accessToken = dataSso.accessToken;

    // Provide text data as input prompt
    data = {"prompt": "What Are Recurrrent Neural Networks?"};

    headers = {
      "accept": "application/json",
      "Content-Type": "application/json"
    };

    // Build the body of POST request
    var body = {
      access_token: accessToken,
      machine_id: "<HASHED_MACHINE_ID>", // replace with a valid machine_id
      endpoint: "<ENDPOINT_NAME>", // replace with a valid endpoint_name
      params: data
    };

    // Call On-demand Inference API on LLM deployed on ClearML
    fetch('https://www.snap4city.org/clearml/serve/OnDemand/Stable', {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify(body)
    })
    .then(response => {
      if (!response.ok) {
        throw new Error(`Error in response: ${response.status}`);
      }
      return response.json();
    })
    .then(data => {
      alert(data.answer); // Show API response in an alert message box
    })
    .catch(error => {
      const errorMsg = "Error in Serving Auth node: " + error.message;
      console.log(errorMsg, {
        error: error
      });
      $("#outputText").css("color", "#000").val(errorMsg);
    });
  },
  error: function(errorData) {
```

```

        console.log("Error in retrieving Access Token.");
    }
});

```

8.5 Model Monitoring

The monitoring system was designed to allow administrators to view detailed metrics on model operation and resource utilisation. Thanks to **Grafana** (Figure 8-4), it is possible to monitor the progress of inferences, the distribution of predictions and the resource utilisation of the serving machines in real time.

Administrators can also check the request logs via **ClearML Utils**, where each call to the endpoint is tracked and logged, providing complete control over the operations performed (see Chapter 0).

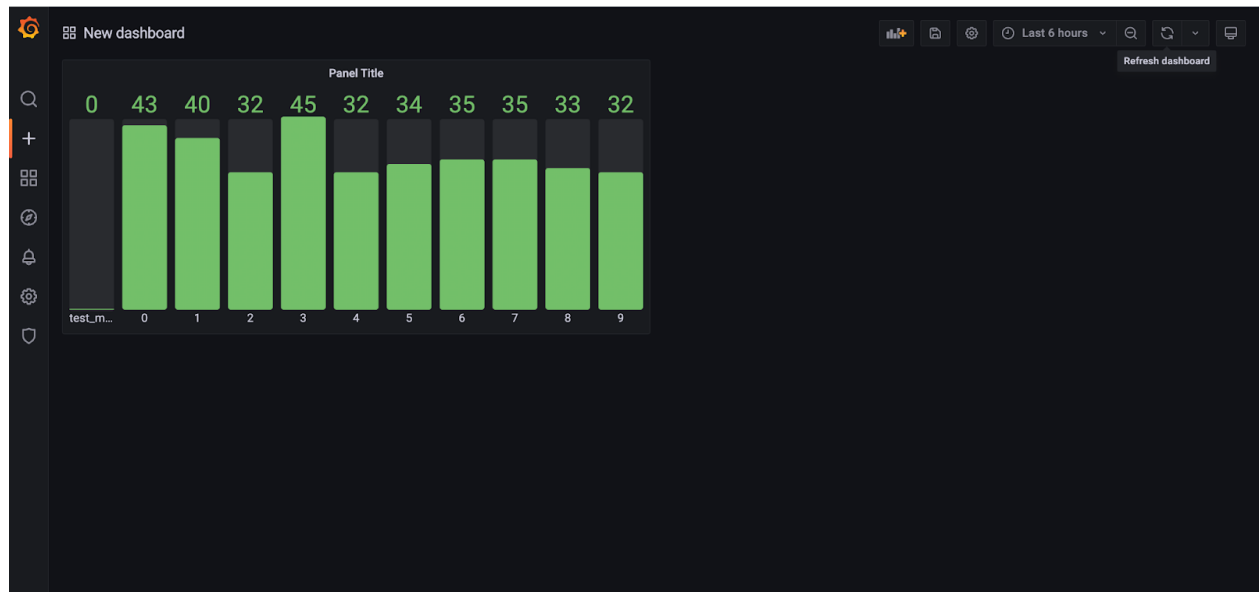


Figure 8-4Test - Model monitoring in production with Grafana, counter of how many times each digit was predicted.

9. Example Scripts

The following Python script provides a simple but comprehensive example showing how to train a Deep Learning model in an integrated ClearML environment. The script exploits Pytorch to build a Neural Network composed by a LSTM and a Feed Forward layer for univariate time-series forecasting. The integration with ClearML follows the guidelines and practices described in this document.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import torch
import torch.nn as nn
from torch.autograd import Variable
import requests
from clearml import Task, Dataset

# Configuration of Environment Variables for ClearML
os.environ['CLEARML_WEB_HOST'] = "http://192.168.1.XXX:8080"
os.environ['CLEARML_API_HOST'] = "http://192.168.1.XXX:8088"
os.environ['CLEARML_FILES_HOST'] = "http://192.168.1.XXX:8081"

# ClearML API Access and Secret Keys
# Replace "<CLEARML_API_KEY>" and "<CLEARML_API_SECRET>" with your own credentials
os.environ['CLEARML_API_ACCESS_KEY'] = "<CLEARML_API_KEY>"
os.environ['CLEARML_API_SECRET_KEY'] = "<CLEARML_API_SECRET>"

# Initialize ClearML Task
task = Task.init(project_name='Time Series Prediction', task_name='AirPassengers LSTM Prediction', output_uri=True)

# Set Base Docker Image (if needed, e.g. to use specific CUDA drivers in the ClearML agent)
task.set_base_docker('nvidia/cuda:12.4.1-base-ubuntu22.04')

# Adding requirements to ClearML, e.g. to use specific Python packages in the ClearML agent
Task.add_requirements("numpy", "1.23.5")

# ClearML Logger
logger = task.get_logger()

dataset_project = 'Time Series Prediction'
dataset_name = 'AirPassengers Dataset'

# Create ClearML dataset (if not present)
try:
    dataset = Dataset.get(dataset_project=dataset_project, dataset_name=dataset_name)
    print("Dataset found on ClearML.")
except Exception as e:
    print("Dataset not found on ClearML. Creating new Dataset...")
    # Create the dataset
    dataset = Dataset.create(dataset_name=dataset_name, dataset_project=dataset_project)

    # Add local CSV file to ClearML dataset
    # CSV file download from Kaggle: https://www.kaggle.com/datasets/rakannimer/air-passengers
    dataset.add_files('Dataset/AirPassengers.csv')

    # Upload the dataset on ClearML
    dataset.upload()
    dataset.finalize()
    dataset_path = dataset.get_local_copy()
```

```

print(f"Dataset created and uploaded at the following path: {dataset_path}")

# Start remote execution on ClearML (replace "<QUEUE_NAME>" with the name of existing queue)
task.execute_remotely(queue_name="<QUEUE_NAME>", exit_process=True)

# Get a copy of the dataset in the ClearML agent remote context
dataset = Dataset.get(dataset_project=dataset_project, dataset_name=dataset_name,
alias="training_dataset")
dataset_path = dataset.get_local_copy()
logger.report_text(f"Dataset loaded at the following path: {dataset_path}")

# Reading data from csv
data = pd.read_csv(os.path.join(dataset_path, 'AirPassengers.csv'))

# Print the first 10 rows of the dataset
logger.report_text("First 10 rows:")
logger.report_text(data.head(10).to_string())

training_set = data.iloc[:, 1:2].values

# Plot dataset and attach the plot on ClearML
plt.figure(figsize=(15, 5))
plt.plot(data['Month'], data['#Passengers'], label='Original Data')
plt.xlabel('Month')
plt.ylabel('Number of Passengers')
plt.title('Number of Monthly Passengers')
plt.xticks(data['Month'][:,12], rotation=45)
plt.legend()
plt.grid(True)
plt.tight_layout()
logger.report_matplotlib_figure(title='Number of Monthly Passengers', series='Original Data',
figure=plt)
plt.close()

# Data normalization
def min_max_normalize(data):
    data_min = np.min(data)
    data_max = np.max(data)
    normalized_data = (data - data_min) / (data_max - data_min)
    return normalized_data, data_min, data_max

def min_max_denormalize(data, data_min, data_max):
    return data * (data_max - data_min) + data_min

normalized_data, data_min, data_max = min_max_normalize(training_set)

# Data preparation for LSTM
def sliding_windows(data, seq_length):
    x = []
    y = []
    for i in range(len(data) - seq_length - 1):
        _x = data[i:(i + seq_length)]
        _y = data[i + seq_length]
        x.append(_x)
        y.append(_y)
    return np.array(x), np.array(y)

seq_length = 4
x, y = sliding_windows(normalized_data, seq_length)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
logger.report_text(f"Using device: {device}")

# Split dataset into training (70%), validation (15%) and test (15%)
train_size = int(len(y) * 0.7)

```

```

val_size = int(len(y) * 0.15)

trainX = torch.Tensor(x[:train_size]).to(device)
trainY = torch.Tensor(y[:train_size]).to(device)

valX = torch.Tensor(x[train_size:train_size + val_size]).to(device)
valY = torch.Tensor(y[train_size:train_size + val_size]).to(device)

testX = torch.Tensor(x[train_size + val_size:]).to(device)
testY = torch.Tensor(y[train_size + val_size:]).to(device)

# Print dataset sizes
logger.report_text(f"Training dataset size: {trainX.shape}")
logger.report_text(f"Validation dataset size: {valX.shape}")
logger.report_text(f"Test dataset size: {testX.shape}")

# LSTM model
class LSTM(nn.Module):
    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM, self).__init__()
        self.num_classes = num_classes
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size)).to(device)
        c_0 = Variable(torch.zeros(self.num_layers, x.size(0), self.hidden_size)).to(device)
        _, (h_out, _) = self.lstm(x, (h_0, c_0))
        h_out = h_out.view(-1, self.hidden_size)
        out = self.fc(h_out)
        return out

# Model parameters
num_epochs = 1000
learning_rate = 0.001
input_size = 1
hidden_size = 32
num_layers = 1
num_classes = 1

task.connect_configuration({
    'num_epochs': num_epochs,
    'learning_rate': learning_rate,
    'input_size': input_size,
    'hidden_size': hidden_size,
    'num_layers': num_layers,
    'num_classes': num_classes,
    'seq_length': seq_length,
})

lstm = LSTM(num_classes, input_size, hidden_size, num_layers).to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(lstm.parameters(), lr=learning_rate)

# Model training and validation
train_losses = []
val_losses = []

for epoch in range(num_epochs):
    lstm.train()
    outputs = lstm(trainX)
    optimizer.zero_grad()
    loss = criterion(outputs, trainY)
    loss.backward()

```

```
optimizer.step()
train_losses.append(loss.item())
logger.report_scalar(title='Loss', series='Train Loss', iteration=epoch,
value=loss.item())

# Validation
lstm.eval()
with torch.no_grad():
    val_outputs = lstm(valX)
    val_loss = criterion(val_outputs, valY)
    val_losses.append(val_loss.item())
    logger.report_scalar(title='Loss', series='Validation Loss', iteration=epoch,
value=val_loss.item())

if epoch % 100 == 0:
    print(f"Epoch: {epoch}, Train Loss: {loss.item():.5f}, Val Loss:
{val_loss.item():.5f}")

# Plotting loss graphs
plt.figure(figsize=(15, 5))
plt.plot(train_losses, label='Training Loss')
plt.plot(val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.tight_layout()
logger.report_matplotlib_figure(title='Training and Validation Loss', series='Loss Curves',
figure=plt)
plt.close()

# Generation and plot of predictions
lstm.eval()
with torch.no_grad():
    test_outputs = lstm(testX)
    data_predict = test_outputs.cpu().numpy().squeeze()
    dataY_plot = testY.cpu().numpy()

# Denormalizing data
data_predict = min_max_denormalize(data_predict, data_min, data_max)
dataY_plot = min_max_denormalize(dataY_plot, data_min, data_max)
all_data = min_max_denormalize(normalized_data, data_min, data_max)

# Create DataFrame for predictions
predictions_df = pd.DataFrame({
    'Original': dataY_plot.flatten(),
    'Predicted': data_predict.flatten()
})

# Save predictions in csv file
predictions_csv_path = 'predictions.csv'
predictions_df.to_csv(predictions_csv_path, index=False)

# Upload csv file as ClearML artifact
task.upload_artifact(name='final_predictions', artifact_object=predictions_csv_path)

# Get time interval for predictions
test_start_index = train_size + val_size + seq_length
predictions_x = range(test_start_index, test_start_index + len(data_predict))

val_test_split_index = train_size + val_size + seq_length - 1

# Plotting results: Predictions VS Ground Truth
plt.figure(figsize=(15, 5))
plt.plot(all_data, label='Original Data', color='blue')
plt.plot(predictions_x, data_predict, label='Predictions', color='orange')
plt.axvline(x=val_test_split_index, color='green', linestyle='--', label='Test Split')
```



```
plt.xlabel('Time')
plt.ylabel('Passengers')
plt.xticks(
    np.arange(0, len(all_data), step=12),
    labels=[f'Year {i+1}' for i in range(len(all_data) // 12)],
    rotation=45
)
plt.title('Time-Series Prediction')
plt.grid(True)
plt.legend()
plt.tight_layout()
logger.report_matplotlib_figure(title='Time-Series Prediction', series='Prediction vs Original', figure=plt)
plt.close()

# Save Model as ClearML artifact
model_path = 'lstm_model.pth'
torch.save(lstm.state_dict(), model_path)
task.upload_artifact(name='trained_model', artifact_object=model_path)
```

The following figures illustrate the output of the script in the ClearML web UI (:

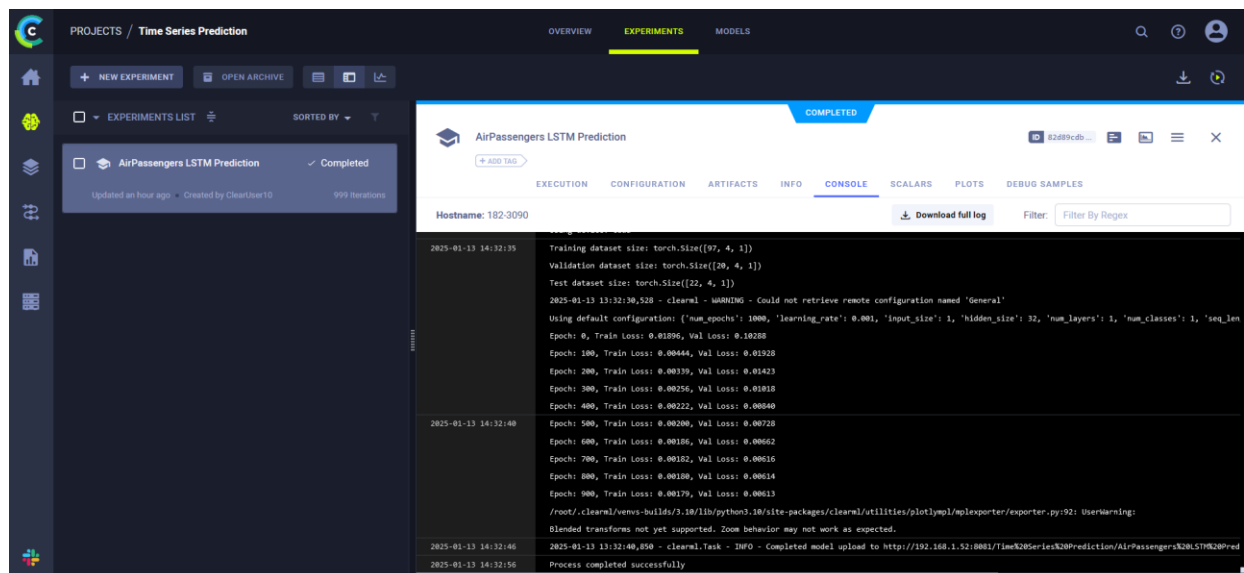


Figure 9-1 ClearML console output for the example script.

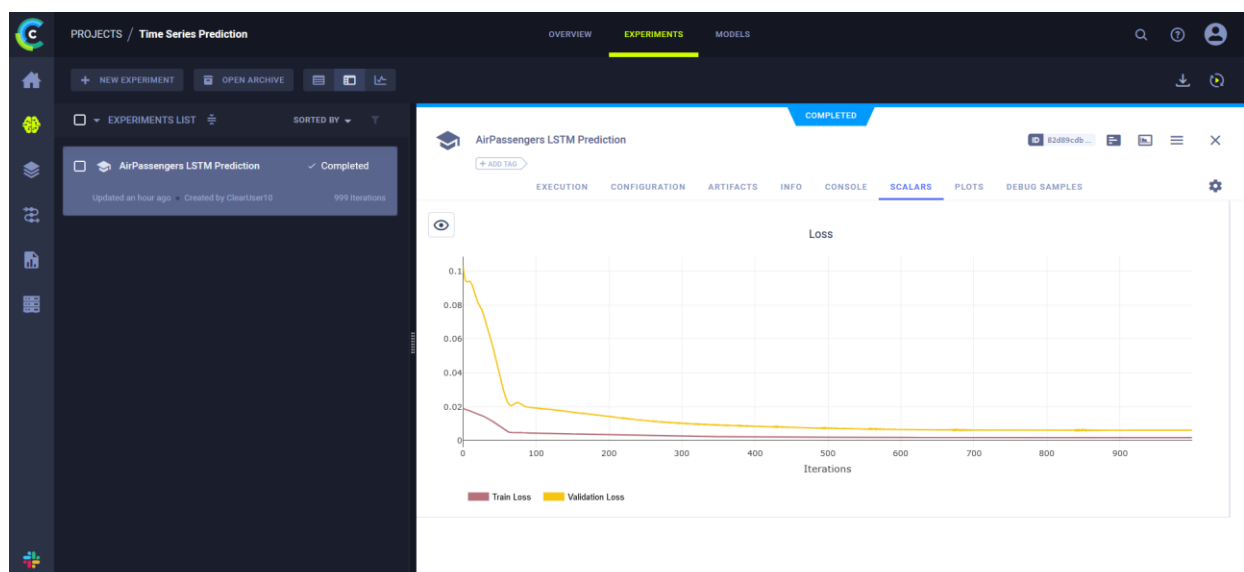


Figure 9-2 ClearML Scalars Tab, showing plots of metrics logged in the example script.

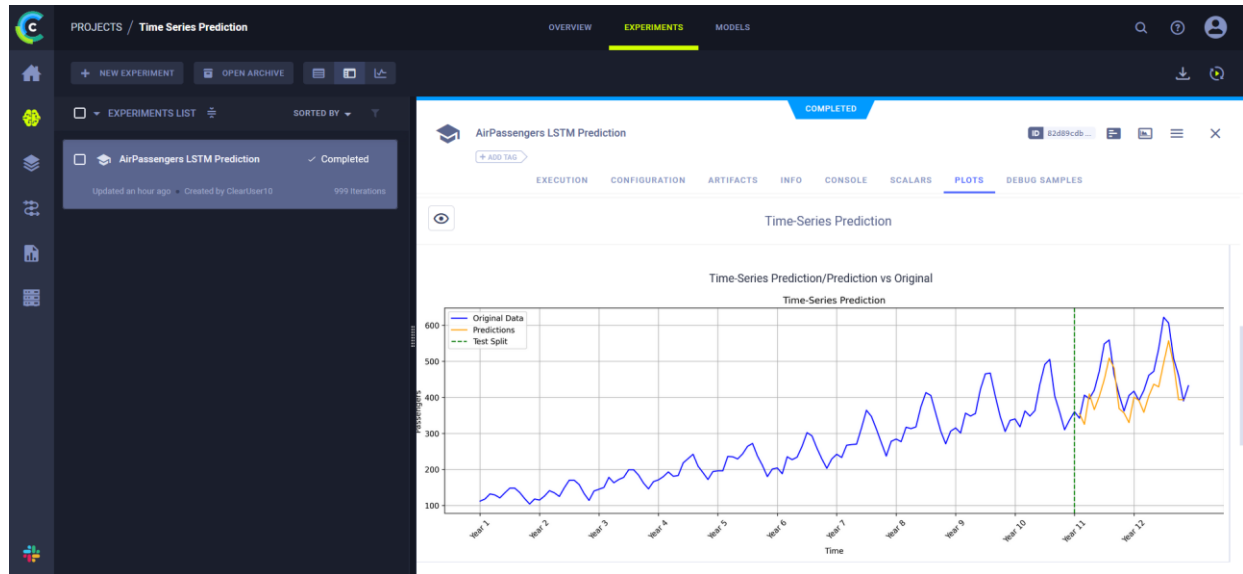


Figure 9-3 ClearML Plots Tab, showing plots attached in the example script.