UNIVERSITÀ DEGLI STUDI FIRENZE

DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB
DISTRIBUTED DATA INTELLIGENCE AND TECHNOLOGIES LAB

https://www.snap4City.org SNAP4CITY KM4CITY

# NeuroSymbolic Artificial Intelligence at Scale


***Paolo Nesi, paolo.nesi@unifi.it***

***Marco Fanfani, marco.fanfani@unifi.it***


***https://www.disit.org/***

**Parte: 0 (2025-26)**



# Introductions

- Adopting MLOps is essential to transforming experimentation from a chaotic process to a **structured workflow**, ensuring that each experiment is **orderly, traceable, and scalable**.

- The heart of this approach lies in the ability to **automatically monitor** metrics, outputs, code, and artifacts, making results not only **repeatable** over time but also **easily shareable** across teams.

- Thanks to an **intuitive interface** for real-time monitoring, resource management becomes extremely practical, allowing to organize each phase of the model lifecycle **efficiently** and **professionally**.

# Why MLOPS?

1. **Calculation and Performance**:
   1. **GPU Orchestration** : Dynamic Task distribuition with remote execution and priority queue
   2. **Bottleneck Optimization** : Workload distribution and Paralle Execution

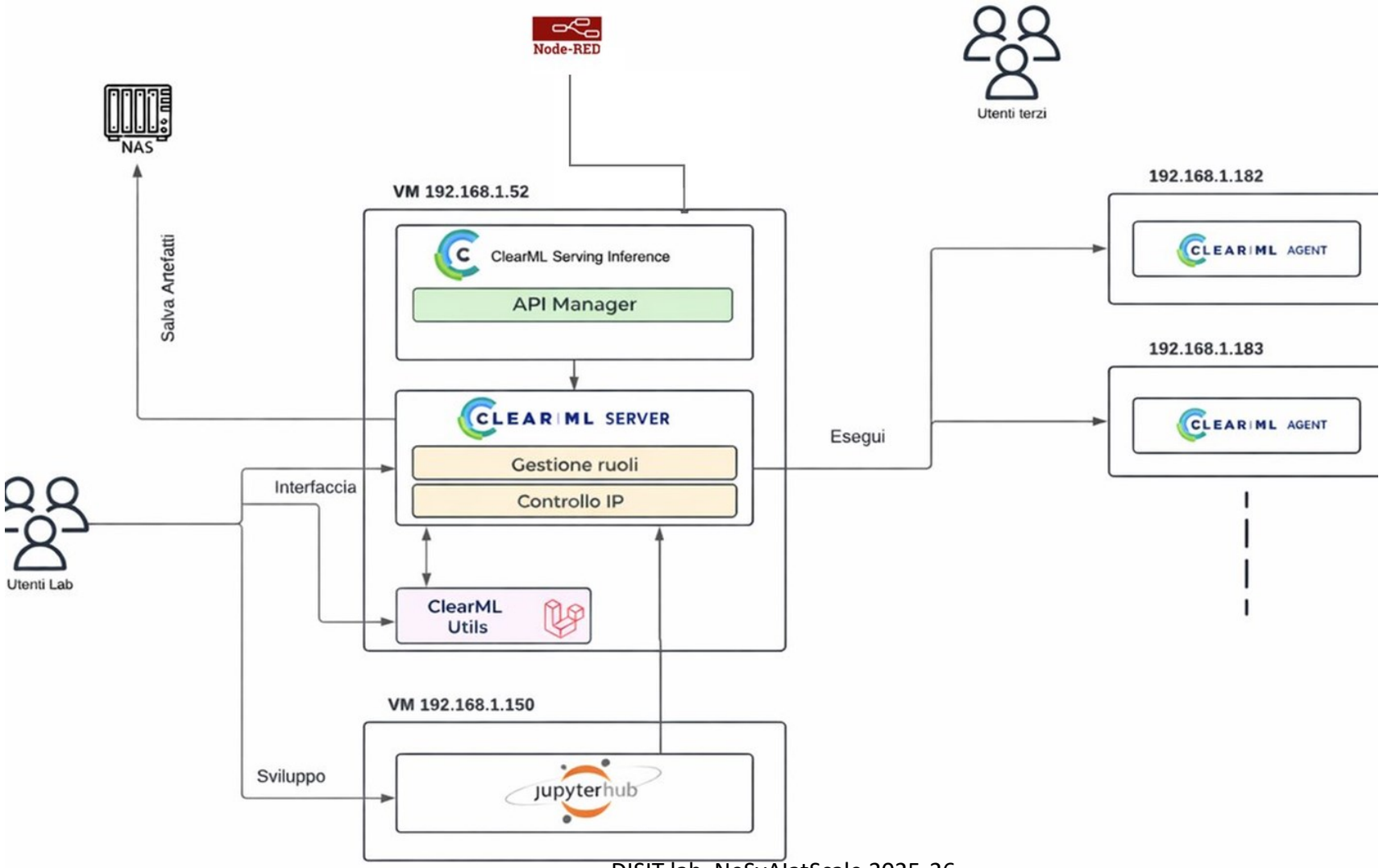
2. **Experiment Management:**
   1. **Experiments Tracking** : The system ensures repeatable and shareable results through automatic monitoring of metrics, code and results.
   2. **Model management** : Facilitates versioning, sharing and archiving of datasets and models.

3. **User experiences**
   1. **Control Interface** : An intuitive interface for real-time monitoring of experiments and resources, designed to organize experiments quickly and efficiently.
   2. **Integration and Compatibility**: It supports many development environments, thus offering a wide variety of workflows and technology stacks.

Section I

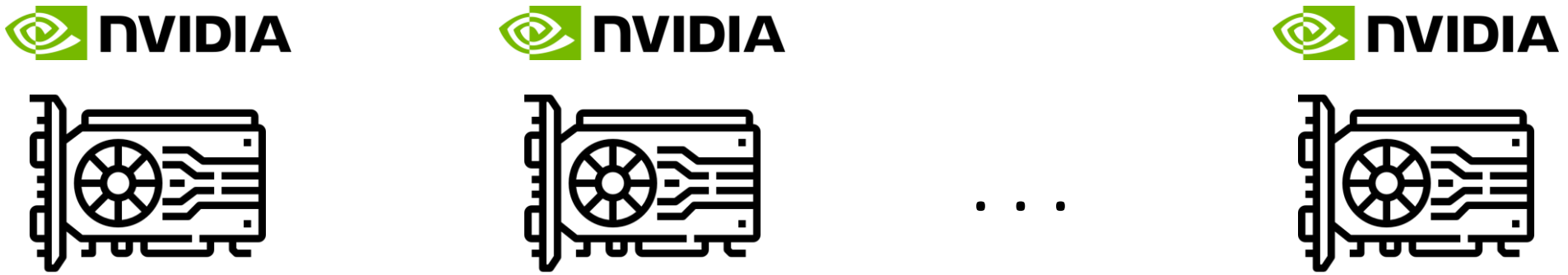
# CLEARML ARCHITECTURE AT SNAP4CITY

# ClearML - Snap4City Architecture

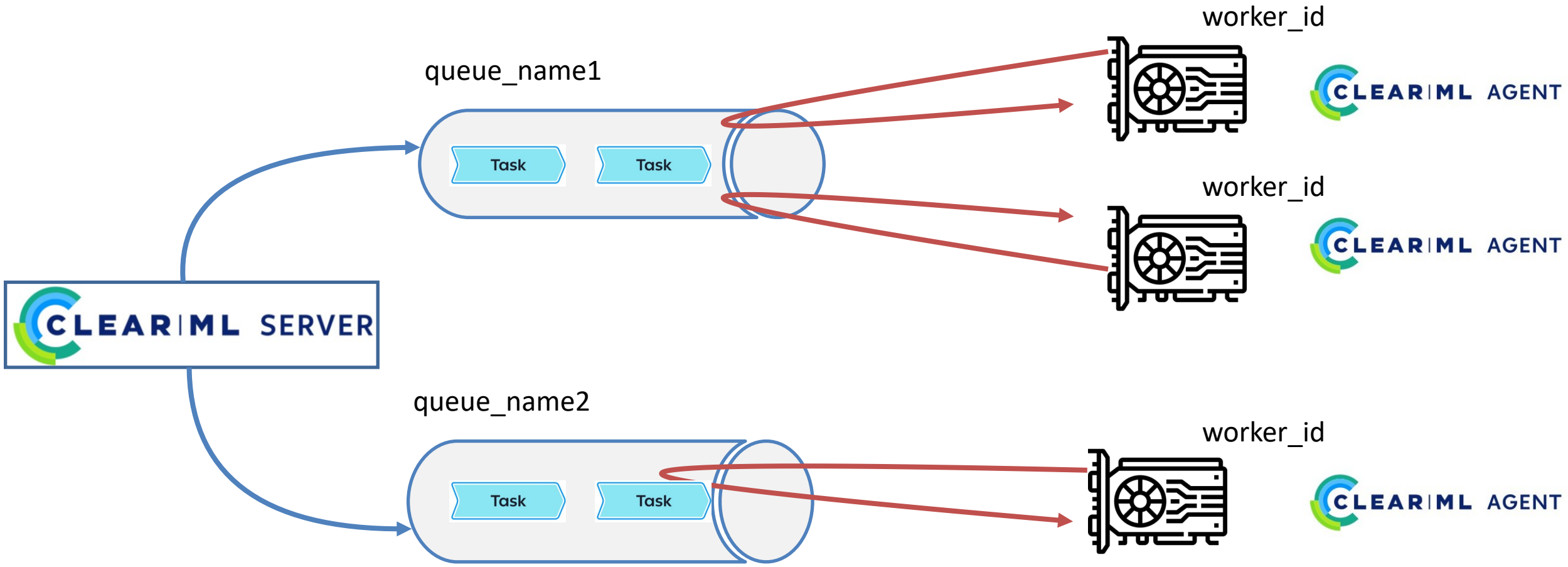


# ClearML Agent

A **ClearML Agent** instance is installed on each of the **Snap4City cluster's GPUs**. This enables task execution and ensures workload distribution.

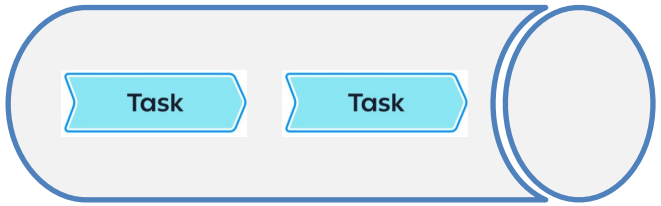


# Workers & Queues



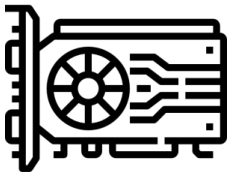
# Agent, Worker and Queue execution

**queue_name**



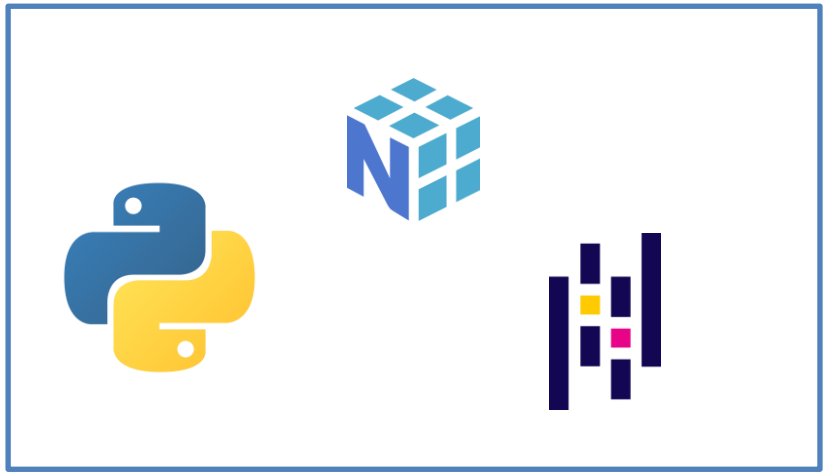


**worker_id**



CLEARML AGENT


docker



CLEARML SERVER

# Roles definition and security management

## General Roles ClearML Setup:

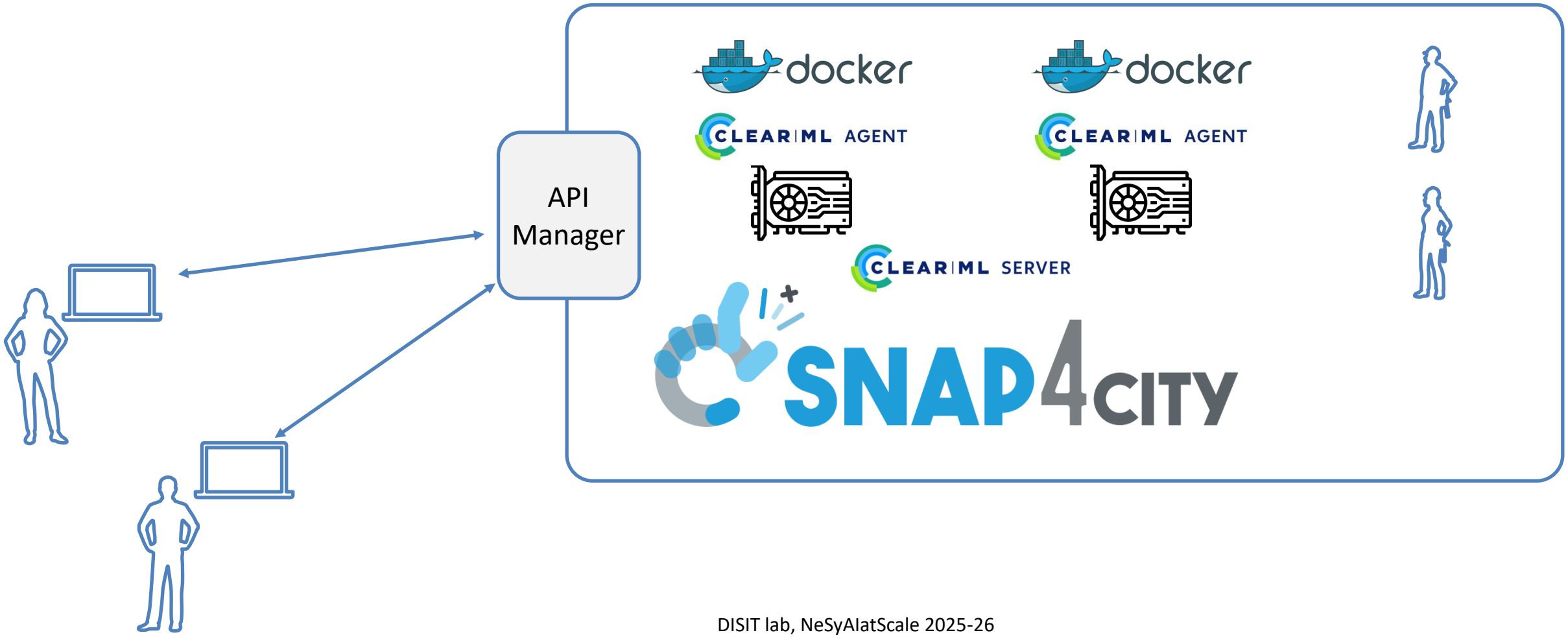
- Allows every user with credentials to create tasks from **all device**.
- Allows every user to **visualize every project** and **visualize queues**.

## Integration of ClearML into the Snap4City Ecosystem :

- Has been designed in order to g**arantee that only verified users** can access to the GPU cluster via IP verification.
- Defined **Admin role** in order to differentiate users with privileges for monitoring purpose and guarantee privacy of common users.
- Live notification system to enusre the **monitoring of task execution** and the **resources availability**.

# What about inference?

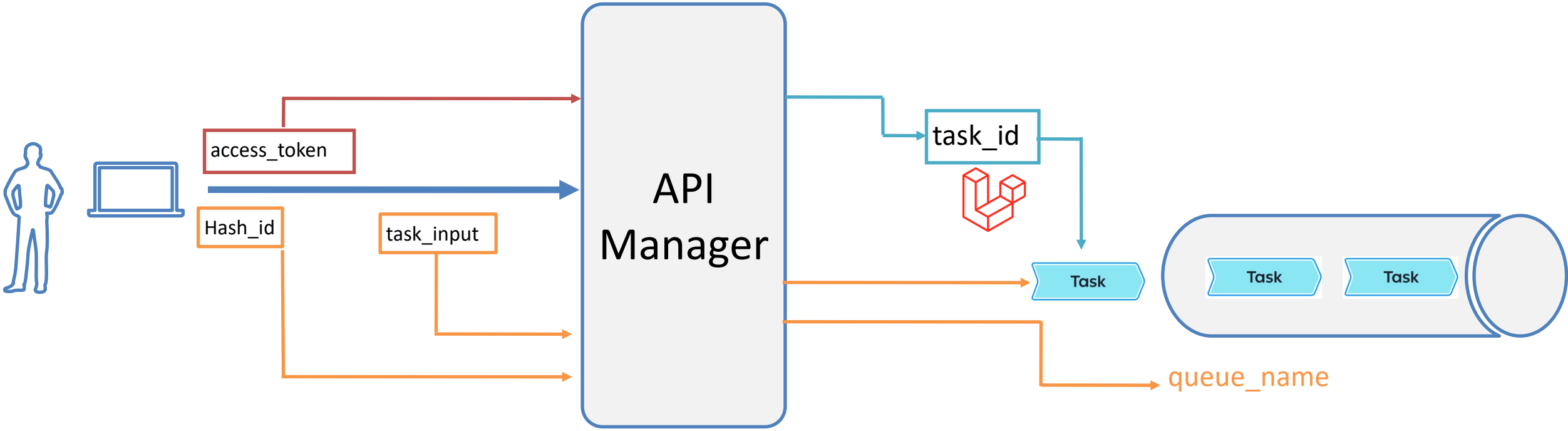
The **verified User** can use the **publish model** for inference via API on ClearML for Snap4City



# Providing services

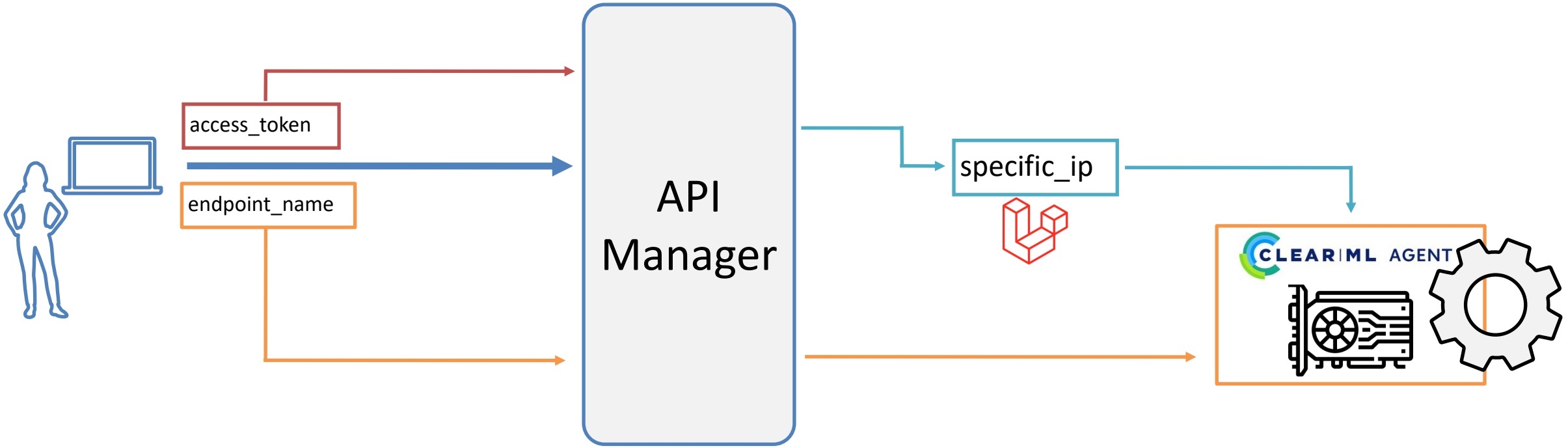
Service provision is enabled in two ways:

- **SPORADIC services for REST calls (API Task Enqueue):** The execution system dynamically allocates temporary containers only at API call time, optimizing resources since memory is not permanently occupied. This is the ideal approach for sporadic tasks where load time is acceptable compared to processing.



# Providing services

- **STABLE services static allocated REST calls (API On-Demand):** Static allocation keeps models permanently in memory after the first load, eliminating latency on subsequent calls. It's ideal for LLM and large-scale models, where high load times would make sporadic mode inefficient.



# ClearML Utils

Developed for internal use only, allows to:

- **Resource management** for API service and task queuing.
- View **available resources** and their **current usage**.
- Visualization tool for monitoring the **progress** of on-demand and **queued requests.**
- Service call monitoring and **log tracking/viewing**

ClearML Utils

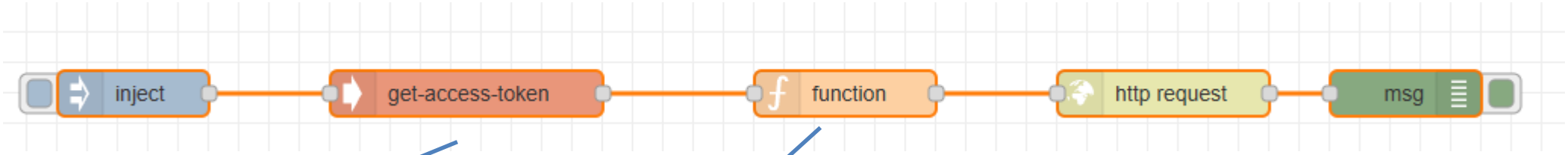
# Node-RED Microservices

Programming **microservices** using **visual programming** with node.js

- **Get access token block**: take the personal access token from Snap4City
- **HTTP Reques block**: call the endpoint for inference using json

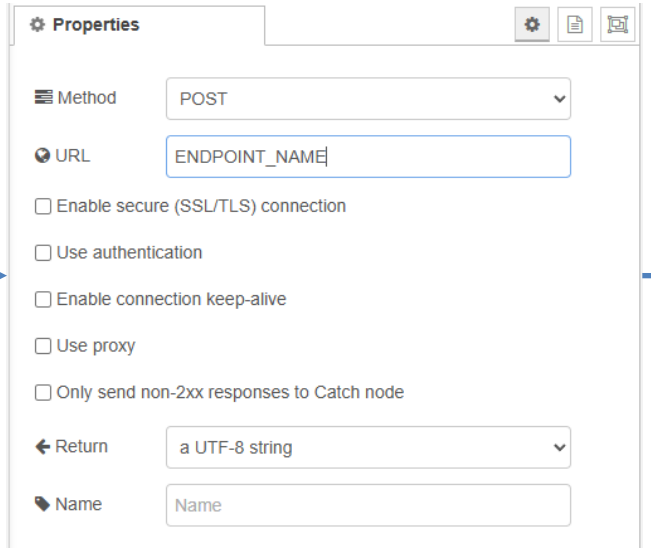
Node-RED

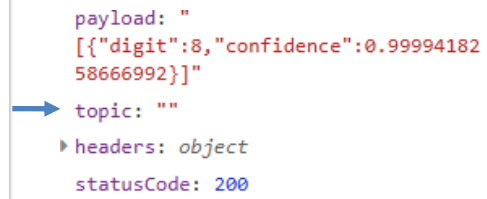
# Node-RED: Flow




Access token saved in msg.payload

```
var access_token = msg.payload;
msg.headers={}
msg.headers["content-type"]= "application/json";
msg.headers["Accept"]= "application/json";
msg.headers["Authorization"]= "Bearer " + access_token;


msg.payload= {
    "access_token": access_token,
    "endpoint": "<ENDPOINT_NAME>",
    "params":{
        "param_1": "value",
    }
}
return msg;
```



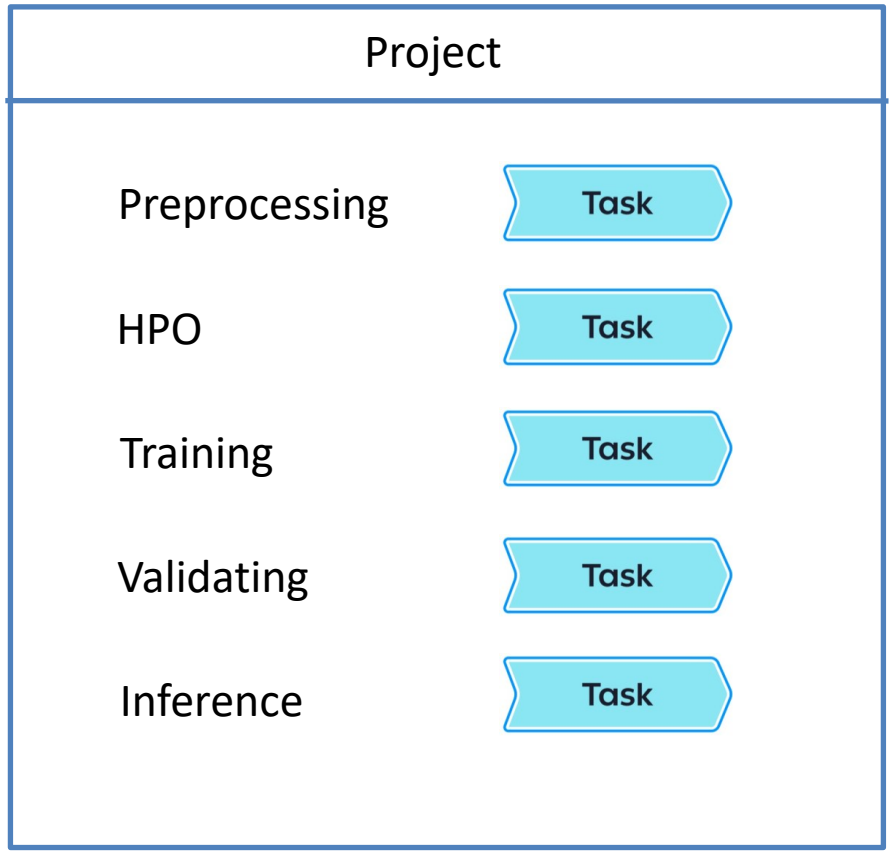



Section II

# CLEARML FUNDAMENTAL CONCEPTS

# Tasks and Projects

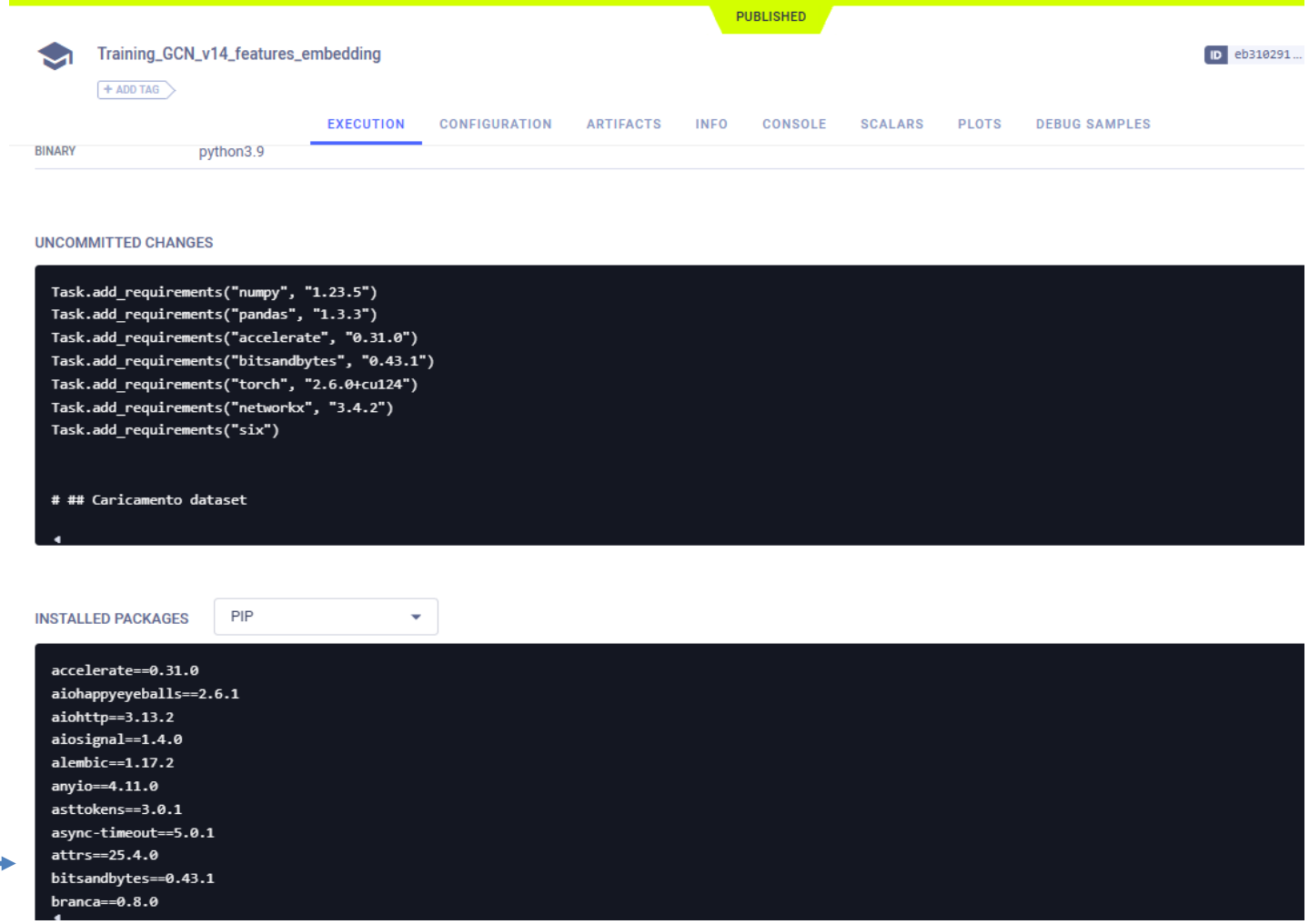
- We refer to Task as the core ClearML component.Can be viewed as an experiment instance.
- It is composed of all that code that we are interested in tracking (a training phase for example). Must be **uniquely identified** and all its code must be recoverable and reproducible.



# Execution

This section is essential for properly **verifying** the running code and installed libraries, It contains:

- The **code** running in the container.
- The **libraries** and their **versions** installed in the container.



# Using JupyterNotebook

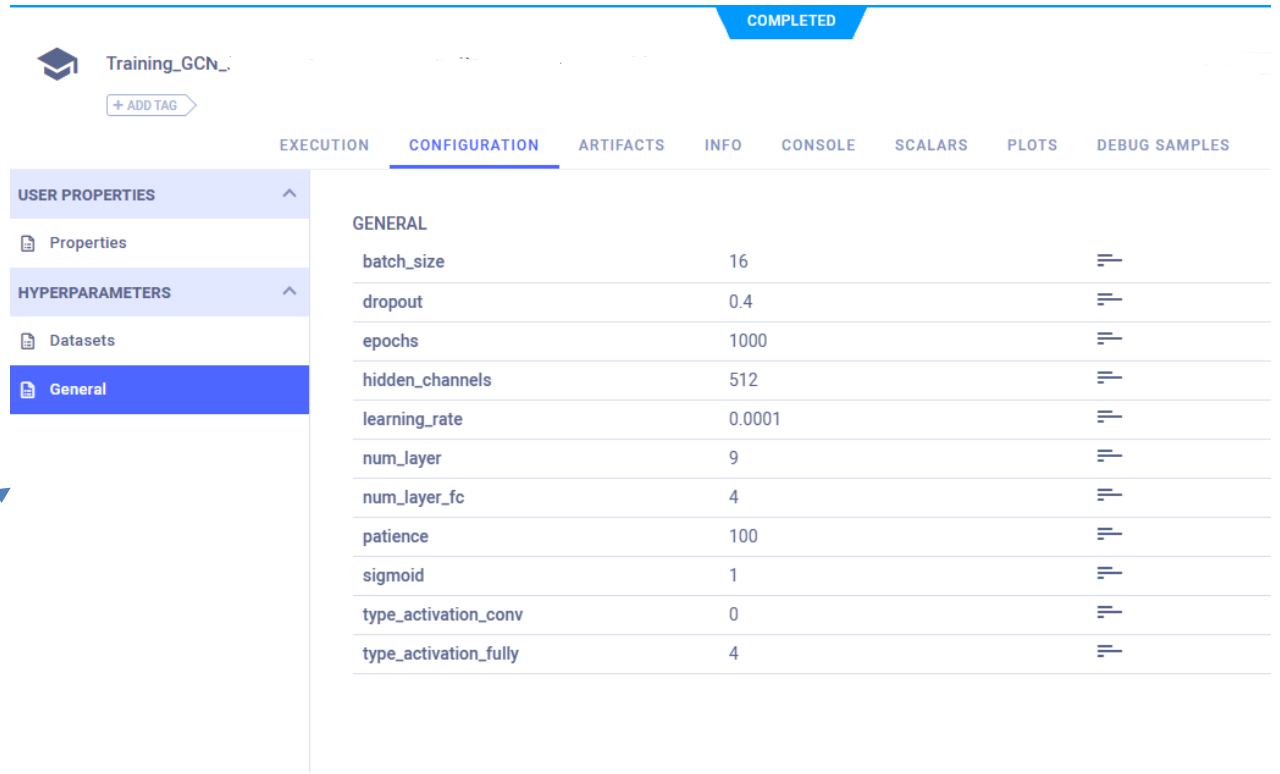
- To implement clean code, functions are obviously used, but passing the task to the function arguments can sometimes lead to problems.
- What you can do is request the task being executed and use the instance created.
- This leads to greater efficiency and also to a greater distinction between functions that interact with the task and functions that do not interact with it.

```
task=Task.current_task()
logger= task.get_logger()
```

# Hyperparameters

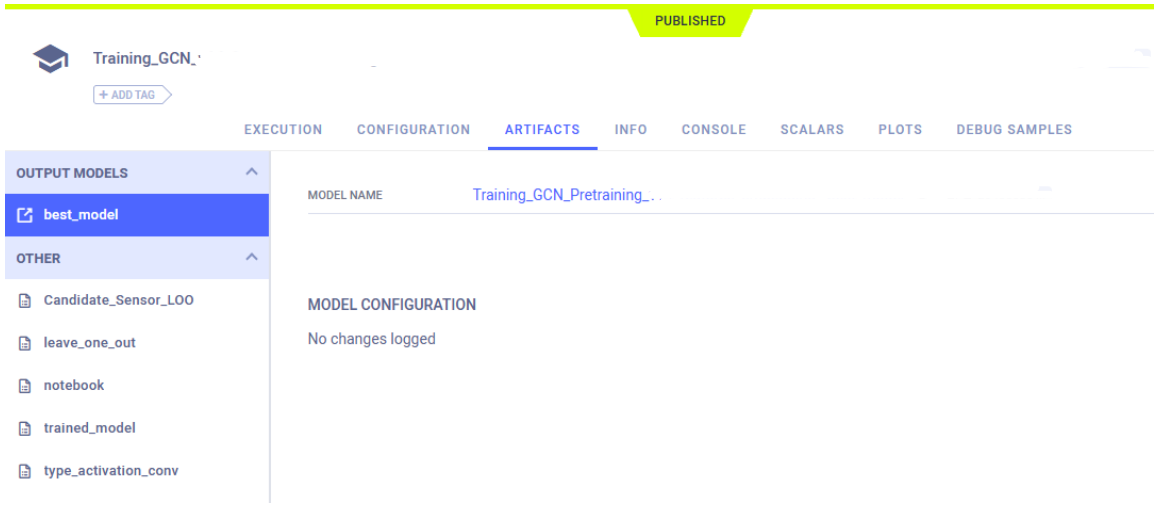
- ClearML provides a **configuration section** that lists the used **datasets** and **general configurations**.
- In this section, ClearML, according to the code below, reports the **configuration of the hyperparameters** essential for the **reproducibility** of the experiment and for **verifying** all results associated with those values.

```
hyperparams = {
    'hidden_channels':512,
    'num_layer': 9,
    'num_layer_fc': 4,
    'dropout':  0.4 ,
    'learning_rate': 0.0001,
    'batch_size': 16,
    'epochs': 1000,
    'patience': 100,
    'type_activation_conv':0,
    'type_activation_fully':4,
    'sigmoid':1
}
task.connect(hyperparams)
```



# Artifacts

- Artifacts are central to the execution of a task, as they comprise a **large portion of the code output**.
- Reproducibility refers to ensuring the user can **store, view, and retrieve** this information.
- In the strict context of machine learning, this section is crucial, as the **best model** at the end of training is **exposed as an artifact**.




**Artifacts are essentially the outputs of a task.**

# Scalars, Plots and Debug Sample

- These sections represent the core of **real-time monitoring**, allowing to constantly monitor the execution of active tasks.

- **Scalars**:
  - Reports the utilization of **machine** (CPU) and **GPU** resources
  - **Time series data**. The X-axis is always a **sequential number**, usually iterations, but can also be an **epoch** or other (e.g., training loss, validation loss, accuracy).
- **Plots**:
  - General graphs and charts, such as histograms, confusion matrices, line graphs, and custom graphs in two ways:
    - **Automatic reporting** (depending on the library)
    - **Manual reporting**.
- **Debug Samples**:
  - Images, audio, and videos.
  - Can be reported per iteration.

# Logger and Automatic Reporting

- The Clearml Logger class allows you to **save task results**, as seen in the previous slide, using the Task object.

```
logger= task.get_logger()
logger.report_scalar(title='Loss', series='Train Loss', iteration=epoch, value=loss_average)
```

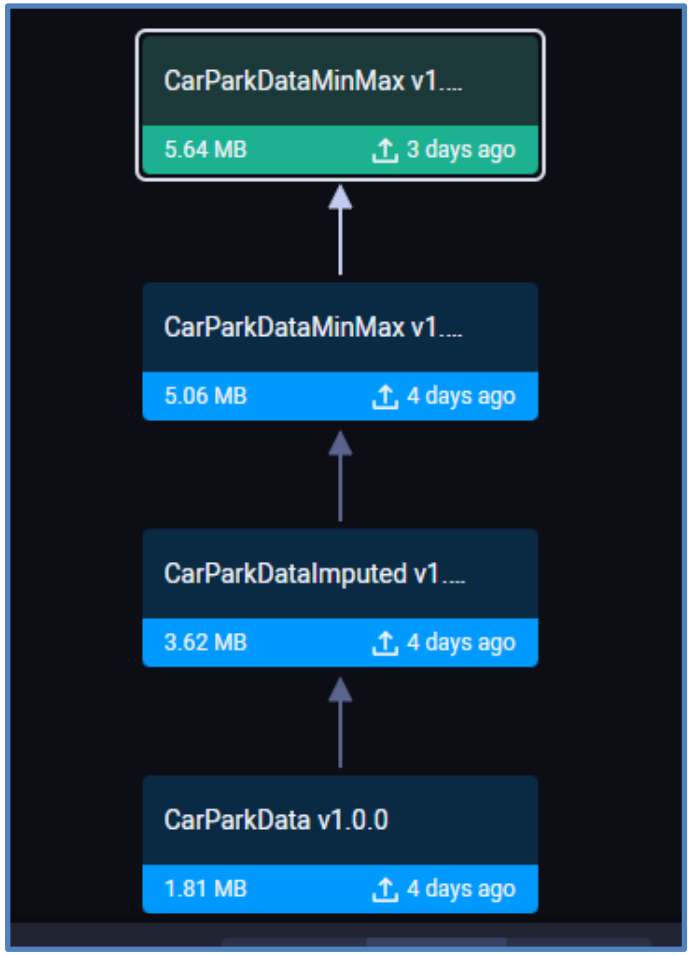
- ClearML **automatically** captures metrics reported to major visualization libraries, such as **TensorBoard** and **Matplotlib**, **without the need to use the Logger** class.

```
plt.figure(figsize=(10,6))
plt.plot(      )
plt.xlabel("   ")
plt.ylabel("   ")
plt.title("   ")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
plt.close()
```

- In addition, ClearML captures and records everything written to **standard output**, from **debug messages** to errors to library **warning messages**.

# Dataset

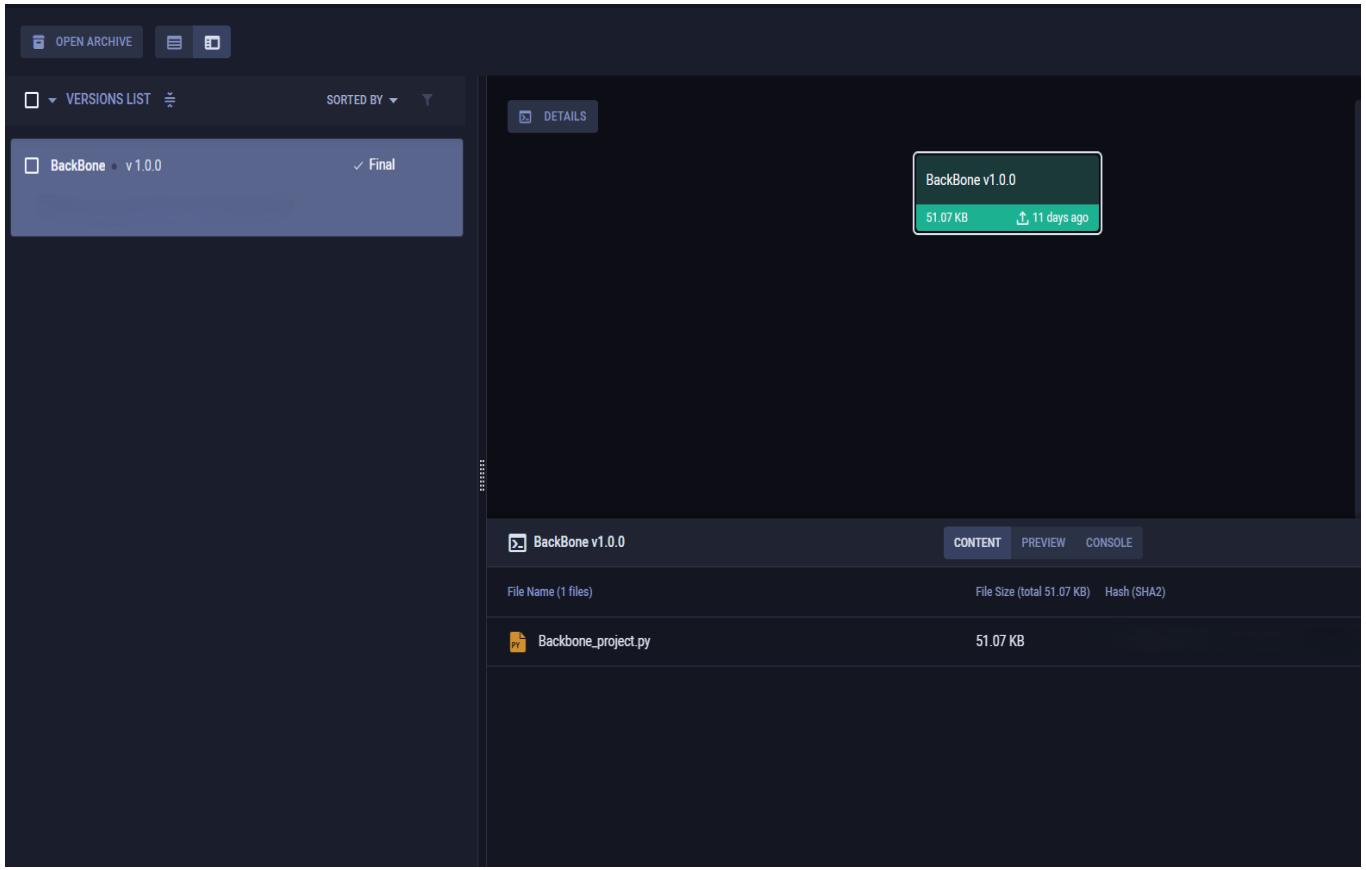
- ClearML allows developers to create, update, upload and retrieve datasets.
- Developers can create **different versions** of the same dataset with incremental approach while **keeping track** of every update.



# Script like Dataset

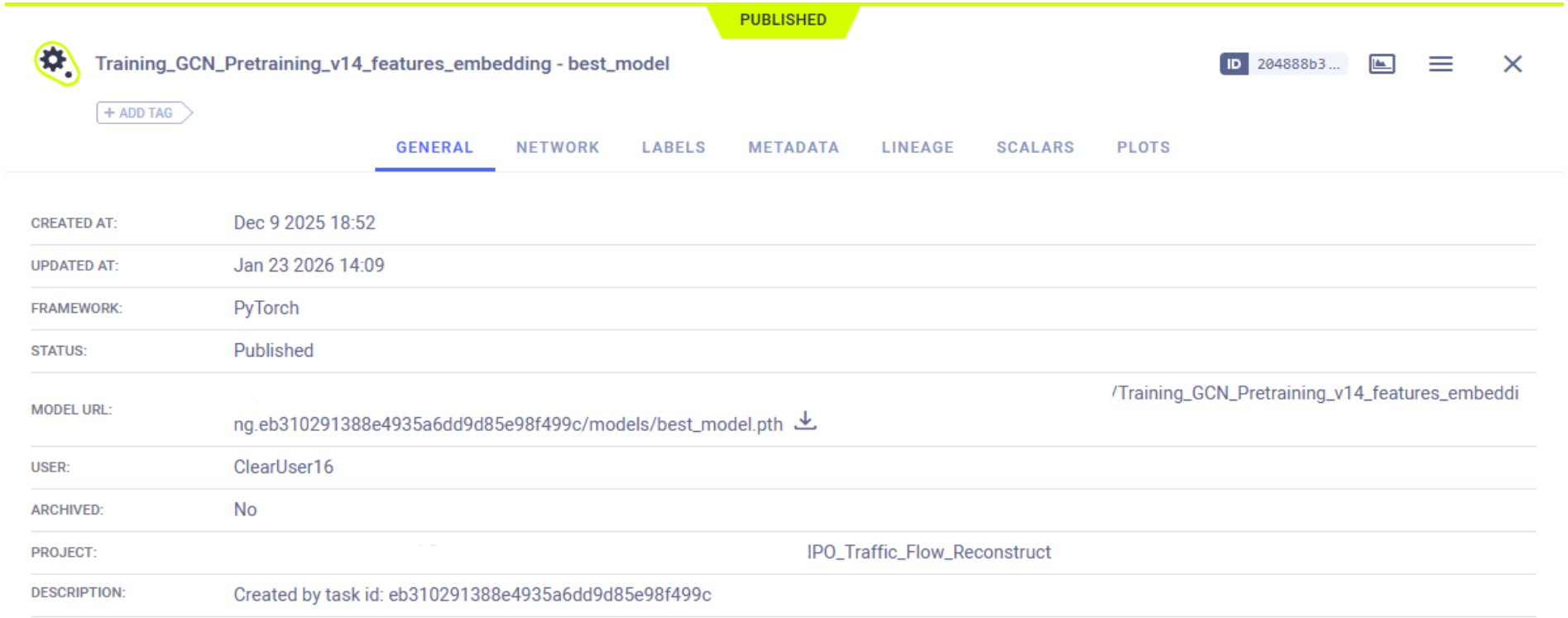
- **Python scripts** are essential for importing and reusing static code and utility modules, helping maintain a clean, modular, and well-organized project structure

- They are treated as real ClearML **datasets**; this allows for dynamic import and updating of custom functions in notebooks



# Model deployment

**Publishing** the model **allows** authorized users to use it for **inference**.



# HPO (Hyperparameter Optimization)

- The goal of HPO is finds the set of hyperparametres values that optimize a specific metric/metrics for the model
- The parameter search space can be specified discrete values or range uniform values
- HPO sampling different parameters set applying a user-selected strategy.
- HPO support different type of optimizer:
  - Optuna: Defualt optimizer, using various sample like grid search, random, Baesyan etc…
  - BOHB: Combines Hyperband searches with orentation and convergence of Baesyan optimization
  - Uniform random: Classic Random search
  - Full grid: Classic Grid search
  - Custom: Custom class by inherting from CLearML automation strategy class

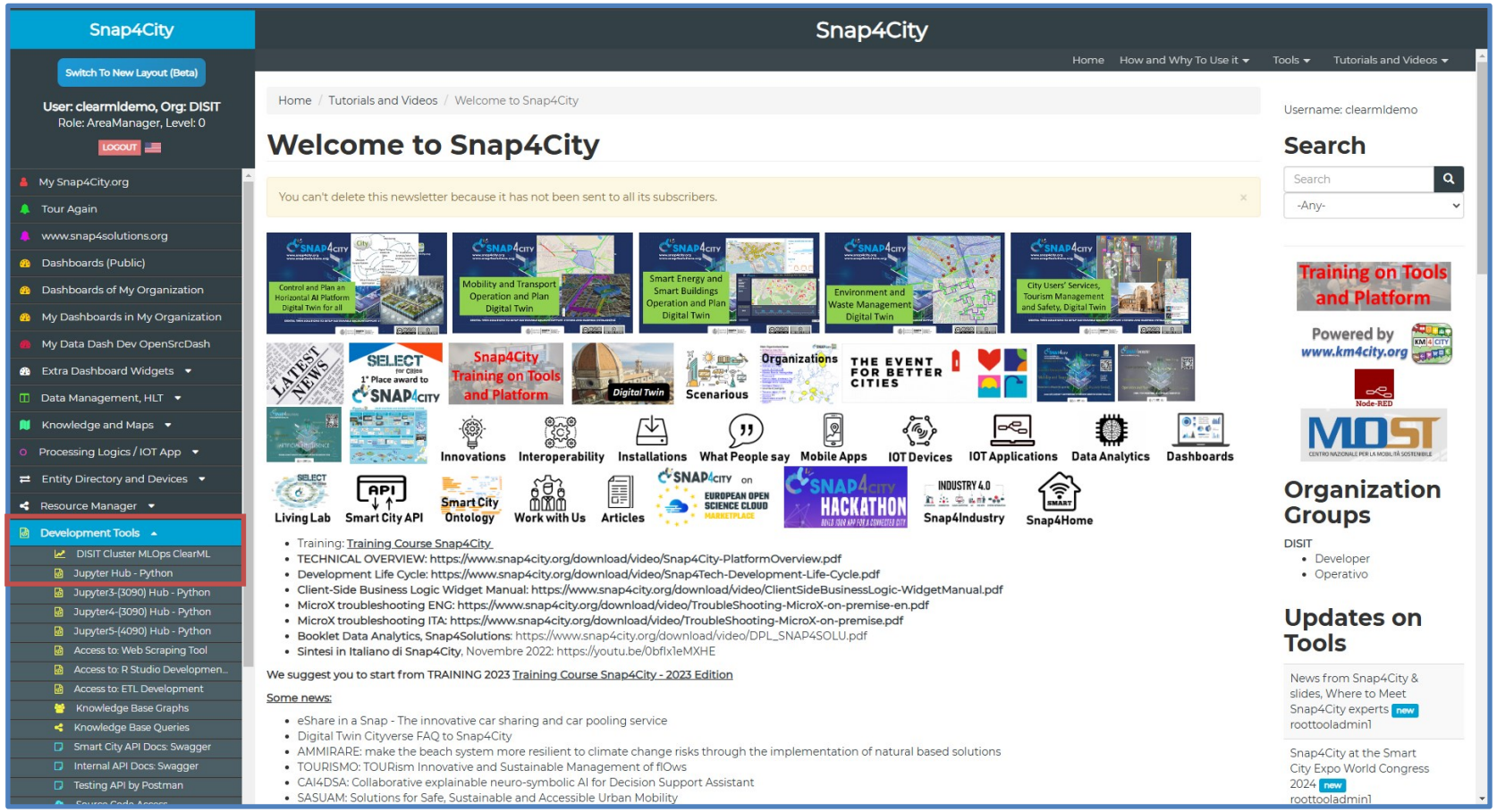
# HPO (Hyperparameter Optimization)

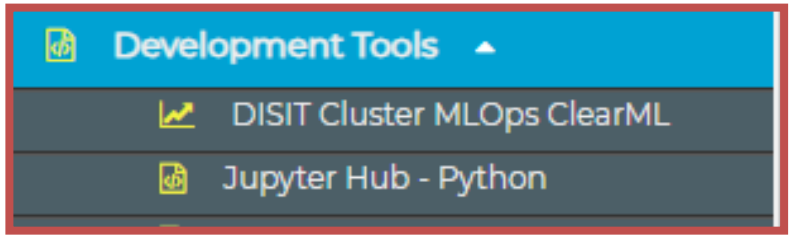
- The structure of HPO is based on hierarchical parent-child task architecture that separate optimization management from training execution.
- Hyperparameter optimization in ClearML is based on a **target task**, which defines the training code and the hyperparameters to be optimized.
- The optimization is managed by a **parent task**, which automatically creates and schedules multiple child tasks, tracks their performance metrics, selects hyperparameters according to the chosen strategy, and ranks the results.
- A key feature is the **separation between management and execution**: the parent task only handles orchestration, while child tasks perform the actual training.
- This enables **distributed execution**, where the parent can run on a local machine and the child tasks are executed on remote GPU workers, ensuring efficient resource utilization and scalable experimentation.

Section III

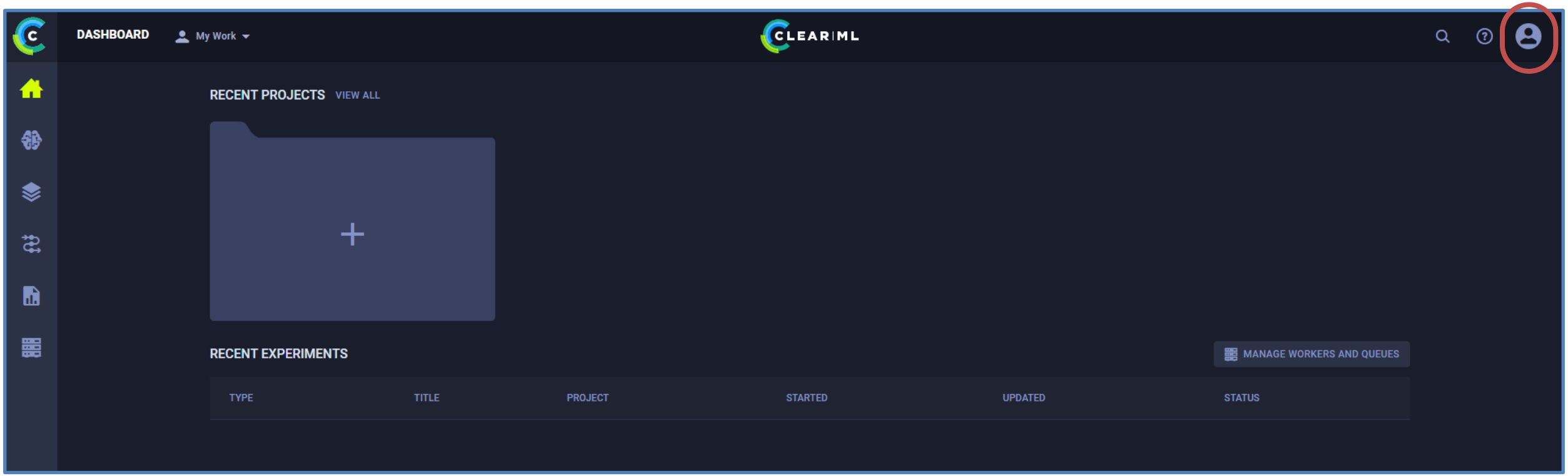
# A GETTING STARTED DEMO

# Authentication with Snap4City

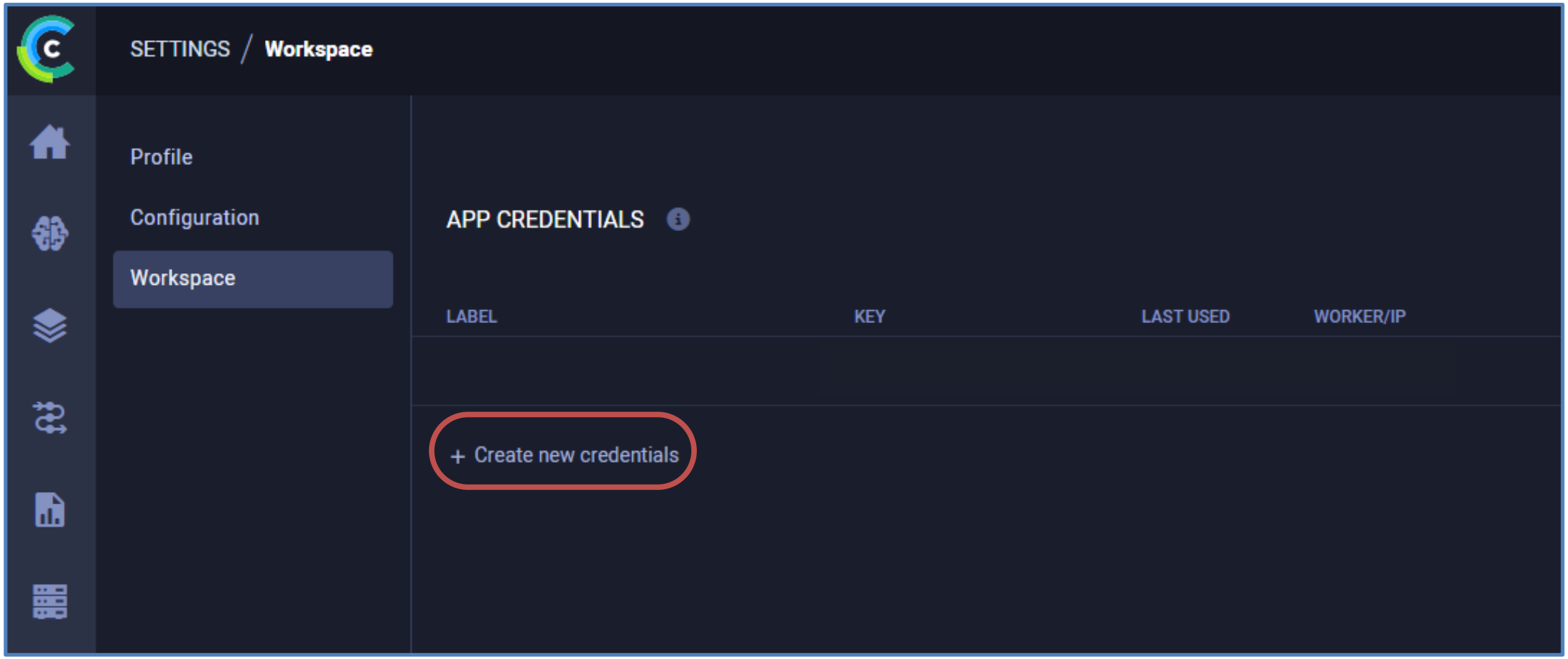


# ClearML Server account



# Getting API keys



# Jupyter notebook: setup

```
import numpy as np
import matplotlib.pyplot as plt
from functools import reduce
import torch
from torchvision.datasets import MNIST
from torch.utils.data import Subset
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as transforms
import os
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score, f1_score,confusion_matrix
from torch.utils.data import DataLoader
from tqdm import tqdm


#----- ClearML Configuration ----------
import clearml
from clearml import Dataset
from clearml import InputModel, Logger, Model, OutputModel,Task
os.environ['CLEARML_WEB_HOST'] = "http://192.168.1.52:8080"
os.environ['CLEARML_API_HOST'] = "http://192.168.1.52:8008"
os.environ['CLEARML_FILES_HOST'] = "http://192.168.1.52:8081"
os.environ['CLEARML_API_ACCESS_KEY'] =
os.environ['CLEARML_API_SECRET_KEY'] =



Task.add_requirements("numpy", "1.23.5")
Task.add_requirements("pandas", "1.3.3")
Task.add_requirements("accelerate", "0.31.0")
Task.add_requirements("bitsandbytes", "0.43.1")
Task.add_requirements("torch", "2.6.0+cu124")
Task.add_requirements("networkx", "3.4.2")
Task.add_requirements("six")
```

# Jupyter notebook: dataset

```
dataset= Dataset.create(
        dataset_project="  ", dataset_name= "  "
)
dataset.add_files(path= "es.json")
dataset.upload()
dataset.finalize()
```

```
dataset= Dataset.get(
        dataset_id="  ", alias= "  "
)
folder_path= datatset.get_local_copy()
```

# Jupyter notebook: script

```
try:
    ds = Dataset.get(
        dataset_project=CODE_PROJECT,
        dataset_name=CODE_NAME,
        only_completed=True
    )
    print("Backbone già importato")
except ValueError:
    print("carico Backbone......")
    ds = Dataset.create(
        dataset_project=CODE_PROJECT,
        dataset_name=CODE_NAME
    )
    ds.add_files('Backbone_project.py')
    ds.upload()
    ds.finalize()


code_ds = Dataset.get(
    dataset_project=CODE_PROJECT,
    dataset_name=CODE_NAME,
    only_completed=True
)
code_path = code_ds.get_local_copy()
sys.path.insert(0, code_path)
from Backbone_project import *
```

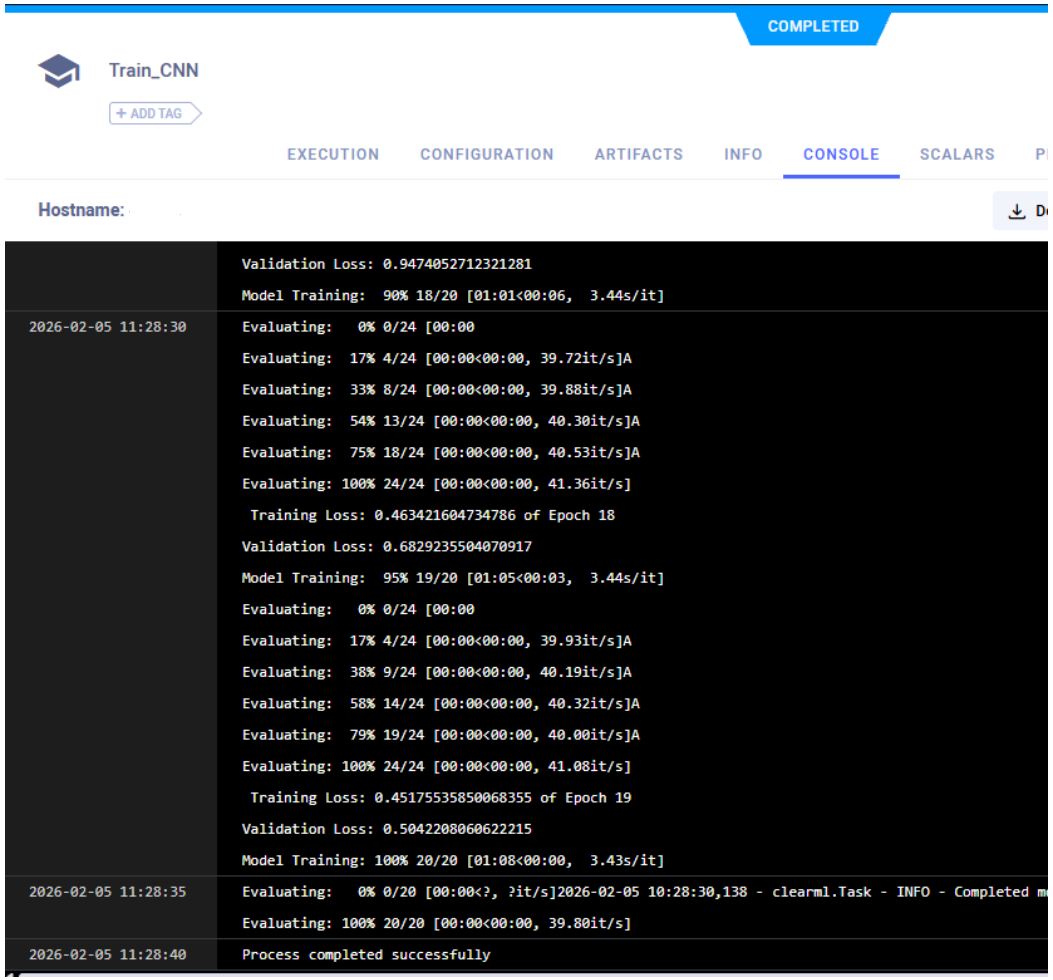
# Jupyter notebook: LifeCycle

Task Initilize

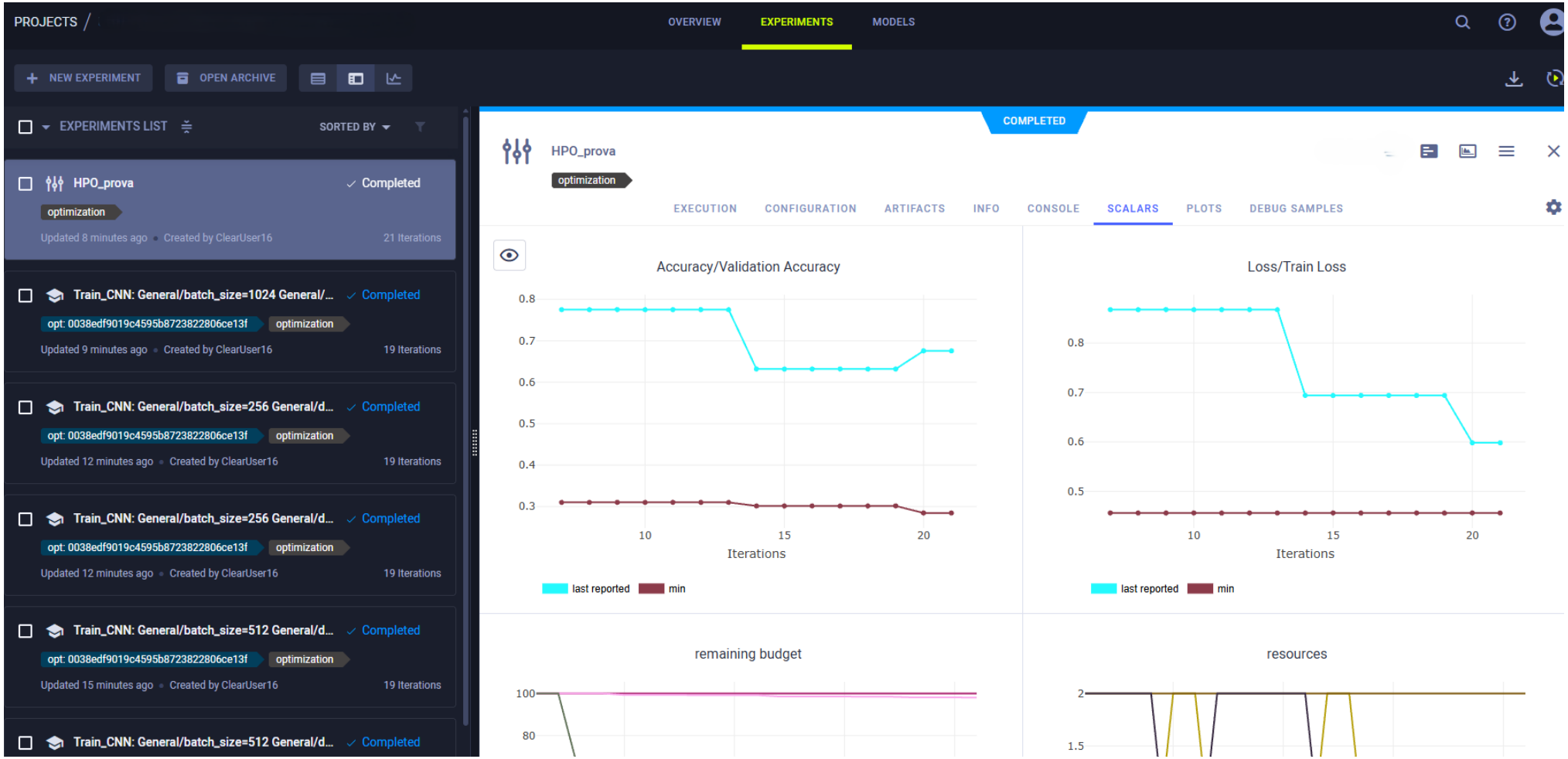
Save Model
Report Metrics
Close task

```python
def train():

    hyperparametres={
        "in_channels": 1,
        "out_channels": 64,
        "depth": 6,
        "num_classes": 10,
        "epochs": 20,
        "batch_size" : 512,
        "learning_rate" : 0.001,
        "weight_decay" : 0.001
    }
    task = Task.init(project_name=f'{NAME_USER}/Lesson_CNN',
                task_name=f'Train_CNN',
                task_type=Task.TaskTypes.training,
                output_uri=True,
                reuse_last_task_id=False,
                continue_last_task=False,
    )

    output_model = OutputModel(task = task, framework = "PyTorch")
    task.set_base_docker("                    ")
    task.execute_remotely(queue_name="          ",exit_process=True)
    task.connect(hyperparametres)
    mnist_ds = Dataset.get(dataset_id="              ")
    local_path = mnist_ds.get_local_copy()
    ds_train, ds_test = Load_Data(local_path)
    device = "cuda" if torch.cuda.is_available() else "cpu"
    logger= task.get_logger()

    model= CNN_Customize(depth=hyperparametres["depth"],
                         in_channels=hyperparametres["in_channels"],
                         out_channels=hyperparametres["out_channels"],
                         num_classes=hyperparametres["num_classes"]
                        )
    model= model.to(device)
    model_trained=Training_Model(model, ds_train, device,hyperparametres["epochs"],hyperparametres["batch_size"],
                  hyperparametres["learning_rate"],hyperparametres["weight_decay"] )

    torch.save(model_trained.state_dict(), "Best_model.pth")

    accurancy_test, report_dict_test, cm, _= Validation_Model(model_trained, ds_test, device,512)
    logger.report_scalar(title="metrics",series="accuracy test",value=accurancy_test,iteration=0)
    logger.report_scalar("metrics", "f1_score", value=report_dict_test["macro avg"]["f1-score"], iteration=0)
    logger.report_scalar("metrics", "precision", value=report_dict_test["macro avg"]["precision"], iteration=0)
    logger.report_scalar("metrics", "recall", value=report_dict_test["macro avg"]["recall"], iteration=0)
    logger.report_confusion_matrix("confusion matrix","test",iteration=1, matrix= cm)
    task.mark_completed()
    task.close()
```

# ClearML: monitoring execution

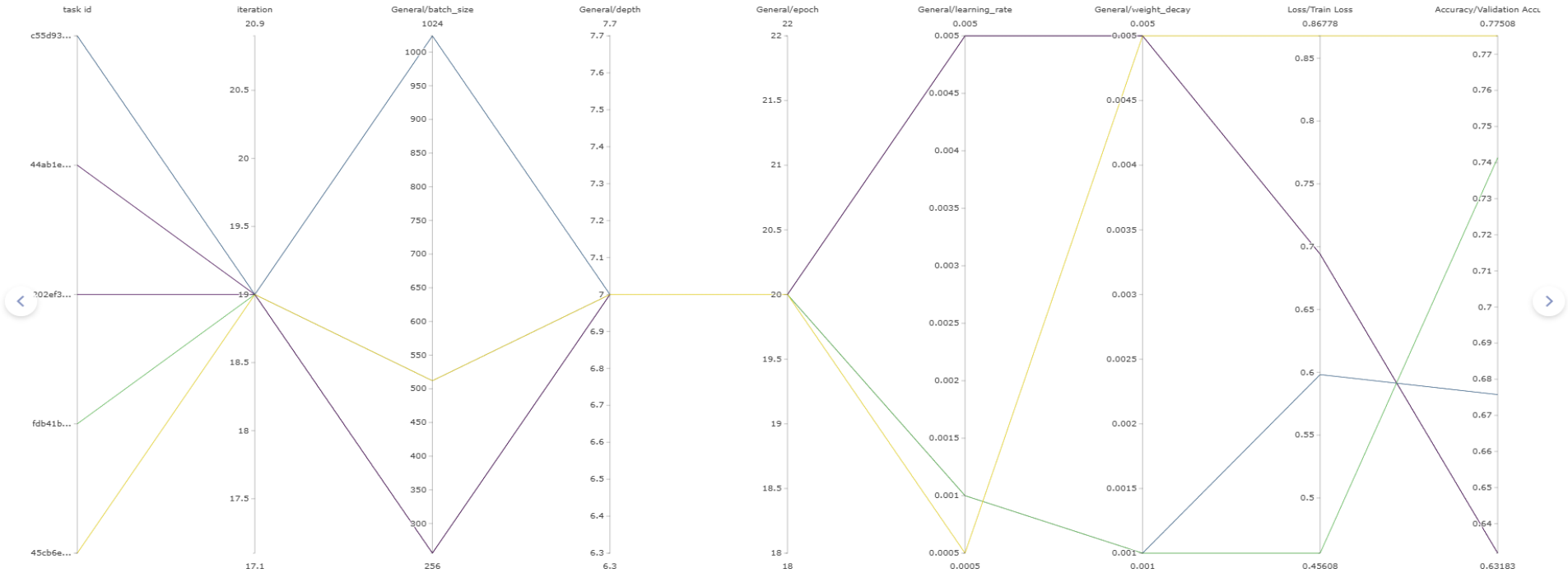



# HPO Example


PROJECTS
OVERVIEW
EXPERIMENTS
MODELS
NEW EXPERIMENT
OPEN ARCHIVE
EXPERIMENTS LIST
SORTED BY
HPO_prova
Completed
optimization
Updated 8 minutes ago • Created by ClearUser16
21 Iterations
Train_CNN: General/batch_size=1024 General/...
Completed
opt: 0038edf9019c4595b8723822806ce13f
optimization
Updated 9 minutes ago • Created by ClearUser16
19 Iterations
Train_CNN: General/batch_size=256 General/d...
Completed
opt: 0038edf9019c4595b8723822806ce13f
optimization
Updated 12 minutes ago • Created by ClearUser16
19 Iterations
Train_CNN: General/batch_size=256 General/d...
Completed
opt: 0038edf9019c4595b8723822806ce13f
optimization
Updated 12 minutes ago • Created by ClearUser16
19 Iterations
Train_CNN: General/batch_size=512 General/d...
Completed
opt: 0038edf9019c4595b8723822806ce13f
optimization
Updated 15 minutes ago • Created by ClearUser16
19 Iterations
Train_CNN: General/batch_size=512 General/d...
Completed
COMPLETED
HPO_prova
optimization
EXECUTION
CONFIGURATION
ARTIFACTS
INFO
CONSOLE
SCALARS
PLOTS
DEBUG SAMPLES
Accuracy/Validation Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
10
15
20
Iterations
last reported
min
Loss/Train Loss
0.8
0.7
0.6
0.5
10
15
20
Iterations
last reported
min
remaining budget
100
80
resources
2
1.5

Parallel Coordinates

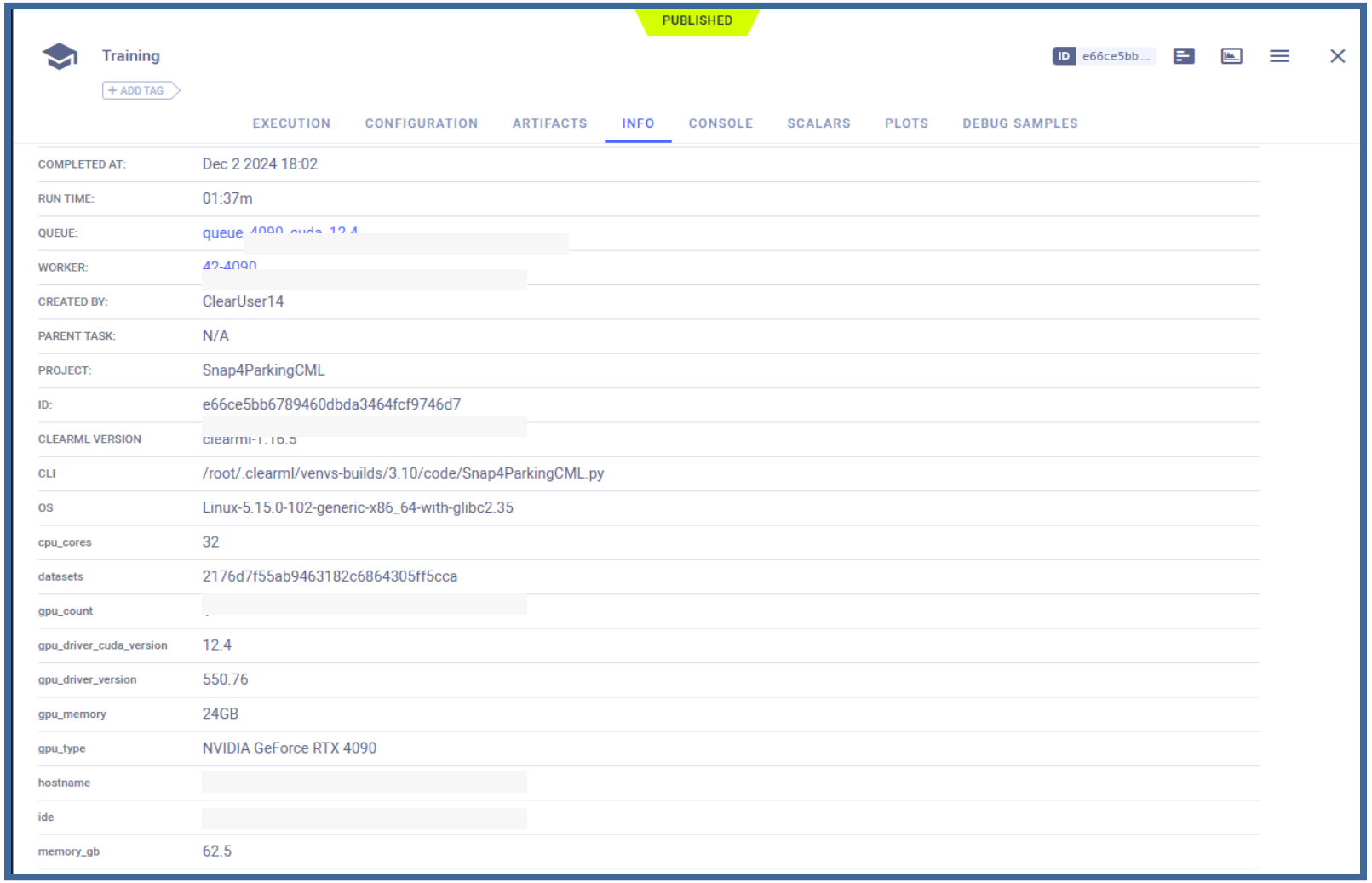
# HPO Example




Summary

| task id | Loss/Train Loss | Accuracy/Validation Accuracy | iteration | General/batch_size | General/depth | General/epoch | General/learning_rate | General/weight_decay | status |
|---|---|---|---|---|---|---|---|---|---|
| 45cb6ee22b8842e597b748 | 0.867783840032334 | 0.7750833333333333 | 19 | 512 | 7 | 20 | 0.0005 | 0.005 | completed |
| fdb41b4a1f39400090cce47 | 0.45607964028703407 | 0.7413333333333333 | 19 | 512 | 7 | 20 | 0.001 | 0.001 | completed |
| 202ef32011624b55995551 | 0.6940597324295247 | 0.6318333333333334 | 19 | 256 | 7 | 20 | 0.005 | 0.005 | completed |
| 44ab1e5ac35746998ca46c | 0.6940597324295247 | 0.6318333333333334 | 19 | 256 | 7 | 20 | 0.005 | 0.005 | completed |
| c55d936668ab49b48ccaa8 | 0.5982046621911069 | 0.67575 | 19 | 1024 | 7 | 20 | 0.001 | 0.001 | completed |

# ClearML: task ending savings 1



# ClearML: task ending savings 2