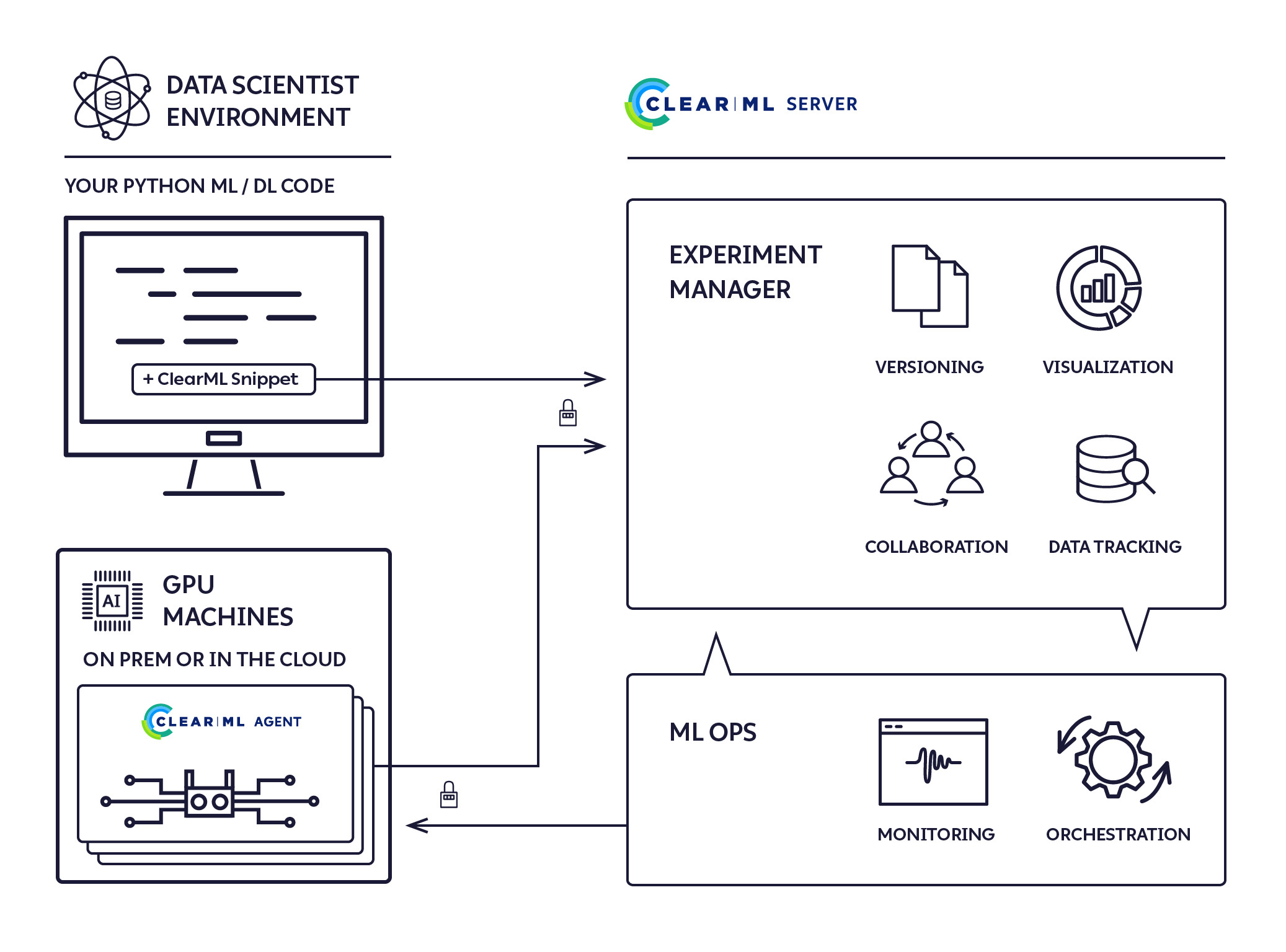
ClearML is an open-source, end-to-end AI Platform designed to streamline AI adoption and the entire development lifecycle. It supports every phase of AI development, from research to production, allowing users to leverage any model, dataset, or architecture at scale [1].



**Before getting start:**

1. Check the deployment of the ClearML server and ClearML agent in VMs at the Openstack project. [If any of the components is missing contact with the admin]

* The ClearML Server is the backend service infrastructure for ClearML. It allows multiple users to collaborate and manage their experiments by working seamlessly with the ClearML Python package and ClearML Agent.
* ClearML Server is composed of the following:
* Web server including the ClearML Web UI, which is the user interface for tracking, comparing, and managing experiments.
* API server which is a RESTful API for:
  + Documenting and logging experiments, including information, statistics, and results.
  + Querying experiments history, logs, and results.
  + File server, which stores media and models making them easily accessible, using the ClearML Web UI.

1. To communicate with ClearML server [2]:

* Create a VM in the Openstack project. (Requirements: Ubuntu 20.04, Flavor: 8cpu-8ram-128disk )
* Create a virtual environment:

sudo apt-get update

sudo apt-get install python3-venv

python3 -m venv myenv

* Activate virtual environment:

source myenv/bin/activate

* Install the clearml python package:

pip install clearml

* Connect ClearML SDK to the Server:
* Execute the following command to run the ClearML setup wizard:

clearml-init

* The setup wizard prompts for ClearML credentials.

Please create new clearml credentials through the settings page in your `clearml-server` web app (e.g. http://localhost:8080//settings/workspace-configuration),

or create a free account at https://app.clear.ml/settings/workspace-configuration

In the settings page, press "Create new credentials", then press "Copy to clipboard".

Paste copied configuration here:

* To get credentials contact with admin
* At the command prompt Paste copied configuration here:, copy and paste the ClearML credentials. The setup wizard verifies the credentials.

Detected credentials key="\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" secret="\*\*\*\*\*\*\*"

CLEARML Hosts configuration:

Web App: https://app.<your-domain>

API: https://api.<your-domain>

File Store: https://files.<your-domain>

Verifying credentials ...

Credentials verified!

New configuration stored in /home/<username>/clearml.conf

CLEARML setup completed successfully.

* Now you are ready to use ClearML into your code!

**Get started to Use ClearML server for data fetching, running experiment and metrics logging:**

* Check InfluxDB credentials:

INFLUXDB\_URL = "……" # InfluxDB URL

TOKEN = "…….." # Your token

ORG = "………" # Your organization name

BUCKET = "…..." # Your bucket name

* To upload a dataset from InfluxDB to ClearML server storage use ClearML Data Management tool. Use the following python code as template to upload a dataset:

from clearml import Dataset

from influxdb import InfluxDBClient

import pandas as pd

import time

import io

# Define the InfluxDB connection parameters

# InfluxDB credentials and configurations

INFLUXDB\_URL = "............" # Replace with your InfluxDB URL

TOKEN = "............." # Your token

ORG = "..........." # Your organization name

BUCKET = ".............." # Your bucket name

# Create InfluxDB client

client = InfluxDBClient(url=INFLUXDB\_URL, token=TOKEN)

# Define the number of days you want to retrieve data from

X\_days = 30 # Change this to the number of days you need

# Calculate the timestamp X days ago from the current time

end\_time = int(time.time() \* 1000000000) # Current time in nanoseconds

start\_time = end\_time - (X\_days \* 24 \* 3600 \* 1000000000) # X days ago in nanoseconds

# Query data from InfluxDB

query = f'SELECT \* FROM "randata" WHERE time >= {start\_time} AND time <= {end\_time}'

result = client.query(query)

# Convert the result to a Pandas DataFrame

df = pd.DataFrame(result.get\_points())

# Convert the DataFrame to a CSV-like string in memory

csv\_buffer = io.StringIO()

df.to\_csv(csv\_buffer, index=False)

csv\_buffer.seek(0) # Reset the buffer to the beginning

# Upload the in-memory CSV data to ClearML

dataset = Dataset.create(dataset\_name="rapp\_data", dataset\_project="rapp\_examples")

dataset.add\_object(csv\_buffer, "randata.csv") # Add the in-memory CSV as an object with a filename

dataset.upload()

dataset.finalize()

print(f"Dataset uploaded to ClearML with ID: {dataset.id}")

* Accessing the uploaded data and prepare it:

from clearml import Dataset

import os

import pandas as pd

from pathlib import Path

# Initialize ClearML Task

from clearml import Task

task = Task.init(project\_name="Your-Project", task\_name="Data Preparation", output\_uri=True)

# Load dataset from ClearML

dataset\_name = "your-dataset-name"

dataset\_project = "your-dataset-project"

local\_dataset\_path = Path(Dataset.get(

dataset\_project=dataset\_project,

dataset\_name=dataset\_name,

alias="dataset-alias"

).get\_local\_copy())

# List all files in the dataset directory

data\_files = [data\_path for data\_path in os.listdir(local\_dataset\_path) if data\_path.endswith(".csv")]

print("Data files:", data\_files)

# Function to preprocess a single CSV file

def process\_file(file\_path):

df = pd.read\_csv(file\_path)

............................. #code for preparing the data

return df

# Process all data files

dataframes = [process\_file(os.path.join(local\_dataset\_path, file)) for file in data\_files]

# Combine into a single DataFrame

combined\_data = pd.concat(dataframes, axis=0).reset\_index(drop=True)

# Display basic stats

print(combined\_data.describe())

* Preprocessing and Normalization:

import os

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, MinMaxScaler

def preprocess\_data(df, numeric\_cols=None, fill\_strategy="mean", drop\_cols=None):

"""

Preprocess the dataset:

- Fill missing values

- Drop unnecessary columns

- Ensure consistent data types

"""

# Drop unnecessary columns if specified

if drop\_cols:

df = df.drop(columns=drop\_cols, errors="ignore")

# Fill missing values

if fill\_strategy == "mean":

df = df.fillna(df.mean(numeric\_only=True))

elif fill\_strategy == "median":

df = df.fillna(df.median(numeric\_only=True))

elif fill\_strategy == "zero":

df = df.fillna(0)

elif fill\_strategy == "ffill":

df = df.fillna(method="ffill")

elif fill\_strategy == "bfill":

df = df.fillna(method="bfill")

else:

raise ValueError("Unsupported fill strategy. Use 'mean', 'median', 'zero', 'ffill', or 'bfill'.")

# Ensure numeric columns are of proper type

if numeric\_cols:

df[numeric\_cols] = df[numeric\_cols].apply(pd.to\_numeric, errors="coerce")

return df

def normalize\_data(df, cols\_to\_normalize, method="standard"):

"""

Normalize the specified columns using the given method:

- 'standard': StandardScaler (z-score normalization)

- 'minmax': MinMaxScaler (scale to [0, 1])

"""

if method == "standard":

scaler = StandardScaler()

elif method == "minmax":

scaler = MinMaxScaler()

else:

raise ValueError("Unsupported normalization method. Use 'standard' or 'minmax'.")

df[cols\_to\_normalize] = scaler.fit\_transform(df[cols\_to\_normalize])

return df

* Creating and Training the model:

import tensorflow as tf

from tensorflow.keras import layers

# Split into training, validation, and test sets

train\_size = int(0.7 \* len(data))

val\_size = int(0.15 \* len(data))

test\_size = len(data) - train\_size - val\_size

train\_data = data[:train\_size]

val\_data = data[train\_size:train\_size+val\_size]

test\_data = data[train\_size+val\_size:]

train\_input, train\_labels = train\_data[:, :-1], train\_data[:, -1]

val\_input, val\_labels = val\_data[:, :-1], val\_data[:, -1]

test\_input, test\_labels = test\_data[:, :-1], test\_data[:, -1]

# Create the model

model = tf.keras.Sequential([

layers.Dense(32, activation='relu', input\_shape=(train\_input.shape[1],)),

layers.Dense(16, activation='relu'),

layers.Dense(8, activation='relu'),

layers.Dense(3, activation='softmax') # Assuming 3 classes

])

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(),

metrics=['accuracy'])

# Train the model

history = model.fit(

train\_input, train\_labels,

validation\_data=(val\_input, val\_labels),

batch\_size=64,

epochs=15,

callbacks=[

tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=5),

tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.1, patience=3)

]

)

print("Training complete!")

* Evaluating and logging matrix:

import matplotlib.pyplot as plt

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(test\_input, test\_labels)

print(f"Test Loss: {test\_loss}, Test Accuracy: {test\_accuracy}")

# Log metrics to ClearML

task.get\_logger().report\_single\_value("Test Loss", test\_loss)

task.get\_logger().report\_single\_value("Test Accuracy", test\_accuracy)

# Log training history

def plot\_training(history):

plt.figure()

plt.plot(history.history['loss'], label='Train Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Loss Curve')

plt.legend()

plt.savefig('loss\_curve.png')

plt.show()

plot\_training(history)

# Save the model

model.save("trained\_model.keras")

task.upload\_artifact("Trained Model", artifact\_object="trained\_model.keras")

[1] <https://clear.ml/docs/latest/docs/>

[2] <https://clear.ml/docs/latest/docs/getting_started/ds/ds_first_steps/#auto-log-experiment>

[3] <https://clear.ml/docs/latest/docs/clearml_data/clearml_data_sdk>