A Cognitive Serverless Framework for the Cloud-Edge Continuum

D4.3 COGNIT Serverless Platform - Scientific Report - c

Version 1.0

11 November 2024

Abstract

COGNIT is an AI-enabled Adaptive Serverless Framework for the Cognitive Cloud-Edge Continuum that enables the seamless, transparent, and trustworthy integration of data processing resources from providers and on-premises data centers in the cloud-edge continuum, and their automatic and intelligent adaptation to optimise where and how data is processed according to application requirements, changes in application demands and behaviour, and the operation of the infrastructure in terms of the main environmental sustainability metrics. This document describes the research and development carried out in WP4 “AI-enabled Distributed Serverless Platform and Workload Orchestration” during the Third Research & Innovation Cycle (M16-M21, according to the new COGNIT architecture, see details in D2.4), providing details on the status of a number of key components of the COGNIT Framework (i.e. Cloud-Edge Manager and AI-Enabled Orchestrator) as well as reporting the work related to supporting Energy Efficiency Optimization in the Multi-Provider Cloud-Edge Continuum.

This project is funded by the European Union’s Horizon Europe research and innovation programme under Grant Agreement 101092711 – SovereignEdge.Cognit

Deliverable Metadata

Project Title:	A Cognitive Serverless Framework for the Cloud-Edge Continuum
Project Acronym:	SovereignEdge.Cognit
Call:	HORIZON-CL4-2022-DATA-01-02
Grant Agreement:	101092711
WP number and Title:	WP4. AI-enabled Distributed Serverless Platform and Workload Orchestration
Nature:	R: Report
Dissemination Level:	PU: Public
Version:	1.0
Contractual Date of Delivery:	30/09/2024
Actual Date of Delivery:	11/11/2024
Lead Author:	Monowar Bhuyan (UMU) & Paul Townend (UMU)
Authors:	Yashwant Sing Patel (UMU), Malik Bouhou (CETIC), Aritz Brosa (Ikerlan), Javad Forough (UMU), Idoia de la Iglesia (Ikerlan), , Fátima Fernández (Ikerlan), Carlos Lopez (ACISA), Tomasz Korniluk (Phoenix), Johan Kristiansson (RISE), Antonio Lalaguna (ACISA), Marco Mancini (OpenNebula), Alberto P. Martí (OpenNebula), Philippe Massonet (CETIC), Nikolaos Matskanis (CETIC), Daniel Olsson (RISE), Mikel Irazola (Ikerlan), Álvaro Puente (Ikerlan), Bruno Rodríguez (OpenNebula), Juan José Ruiz (ACISA), Kaja Swat (Phoenix), Thomas Ohlson Timoudas (RISE), Iván Valdés (Ikerlan), Constantino Vázquez (OpenNebula), David Carracedo (OpenNebula), Ignacio M. Llorente (OpenNebula), Victor Palma (OpenNebula), Michal Opala (OpenNebula), Pavel Czerny (OpenNebula), Jackub Walczak (OpenNebula), Francesco Renzi (Nature 4.0), Riccardo Valentini (Nature 4.0)
Status:	Submitted

Document History

Version	Issue Date	Status[1]	Content and changes
0.1	28/10/2024	Draft	Initial Draft
0.2	05/11/2024	Peer-Reviewed	Reviewed Draft
1.0	11/11/2024	Submitted	Final Version

Peer Review History

Version	Peer Review Date	Reviewed By
0.1	31/10/2024	Idoia de la Iglesia (Ikerlan)
0.1	05/11/2024	Antonio Álvarez (OpenNebula)

Summary of Changes from Previous Versions

First Version of Deliverable D4.3

[1] A deliverable can be in one of these stages: Draft, Peer-Reviewed, Submitted, and Approved.

Executive Summary

This is the third “COGNIT Serverless Platform - Scientific Report” that has been produced in WP4 “AI-enabled Distributed Serverless Platform and Workload Orchestration”. It describes in detail the progress of the software requirements (revised according to the COGNIT Architecture 2.0) that have been active during the Third Research & Innovation Cycle (M16-M21) in connection with these main components of the COGNIT Framework:

Cloud-Edge Manager

- **SR4.3** Serverless Runtime Deployment:

  The Cloud-Edge Manager must be able to deploy Serverless Runtimes as Virtualized Workloads within an Edge Cluster.

- **SR4.4** Metrics, Monitoring, and Auditing:

  Edge-Clusters monitoring, Serverless Runtimes metrics collection and continuous security assessment.

- **SR4.5** Authentication & Authorization:

  Authentication and authorization mechanisms for accessing cloud-edge infrastructure resources by the devices for offloading workloads.

AI-Enabled Orchestrator

- **SR5.1** Building Learning Models:

  Provide AI/ML models based on collected metrics from the Cloud-Edge Manager monitoring service related to Edge Clusters, Serverless Runtimes, and infrastructure usage.

- **SR5.2** Smart management of Cloud-Edge resources:

  AI-Enabled Orchestrator is responsible for managing and optimizing the lifecycle of Edge Clusters and serverless runtimes within Edge Clusters according to the application requirements, infrastructure and virtual resource usage, and energy-aware policies.

This deliverable has been released at the end of the Third Research & Innovation Cycle (M21), and will be updated with incremental releases at the end of each research and innovation cycle in M27, and M33.

Table of Contents

Abbreviations and Acronyms

AE	Auto Encoder
AI	Artificial Intelligence
AI-O	AI-Enabled Orchestrator
API	Application Programming Interface
CLI	Command Line Interface
CMA	Carbon-aware Model Agent
CPCA	Common Principal Component Analysis
DB	Database
DL	Deep Learning
EVPN	Ethernet VPN
FaaS	Function as a Service
FFNN	Feed-Forward Neural Network
GC	Global Controller
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HTTP	Hypertext Transfer Protocol
IDEC	Improved Deep Embedded Clustering
IP	Internet Protocol
IPAM	IP Address Management
JSON	Javascript Object Notation
KVM	Kernel Virtual Machine
LC	Local Controller
LSTM	Long Short-Term Memory
MAPE-K	Monitoring, Analysis, Planning, Execution, and Knowledge
MI	Million Instructions
MIPS	Million Instructions Per Second
ML	Machine Learning
MSE	Mean Squared Error
MTS	Multivariate Time Series
NSGA-II	Non-dominated Sorting Genetic Algorithm II
OS	Operating System
QoS	Quality of Service
RAPL	Running Average Power Limit
REST	Representational State Transfer

RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SLA	Service Level Agreement
SLO	Service Level Objective
SoC	State of Charge
SVD	Single Value Decomposition
TCN	Temporal Convolutional Network
VM	Virtual Machine

1. Cloud-Edge Manager

Cloud-Edge manager is responsible for autonomous management of distributed cloud-edge continuum resources according to the application demand and availability of resources. This development cycle has made progress related to three software requirements, namely, SR4.3 Serverless Runtime Deployment, SR4.5 Authentication & Authorization. Details of each software requirement are reported in Deliverable D2.4.

1.1 [SR4.3] Serverless Runtime Deployment

1.1.1 Cross-Site Live Migration

The cross-site live migration of serverless runtimes is key in order to avoid interruption in executing functions offload from the Devices.

Using OpenNebula OneProvision, the Cloud/Edge Manager frontend (implemented by an OpenNebula frontend) can provision hypervisor nodes on different regions of several cloud providers. When performing this operation, two OpenNebula clusters will be created, each with its own set of datastores, networking and hypervisor hosts.

The networking created for each OpenNebula cluster has two components:

- **Public network**: This public facing network is implemented using the OpenNebula IPAM driver under the hood. Once set up, it allows, through host natting, to have VMs with public IPs from a requested range.
- **Private network**: A network template that can be used to create a VXLAN private network in EVPN mode. This private network creates an overlay between the provision of KVM hosts.

The VMs deployed on each provision do not share a physical network, so they cannot be migrated between KVM hosts existing in different provisions. To enable this, a private network using VXLAN with EVPN mode needs to be created. This network differs from the one created as a virtual network template when issuing the provision. EVPN is required because multicast traffic is not permitted over the Internet, which is the physical network that connects the KVM hosts from different provisions. A high level view of the network deployment to allow cross site migration is presented in Figure 1.1.

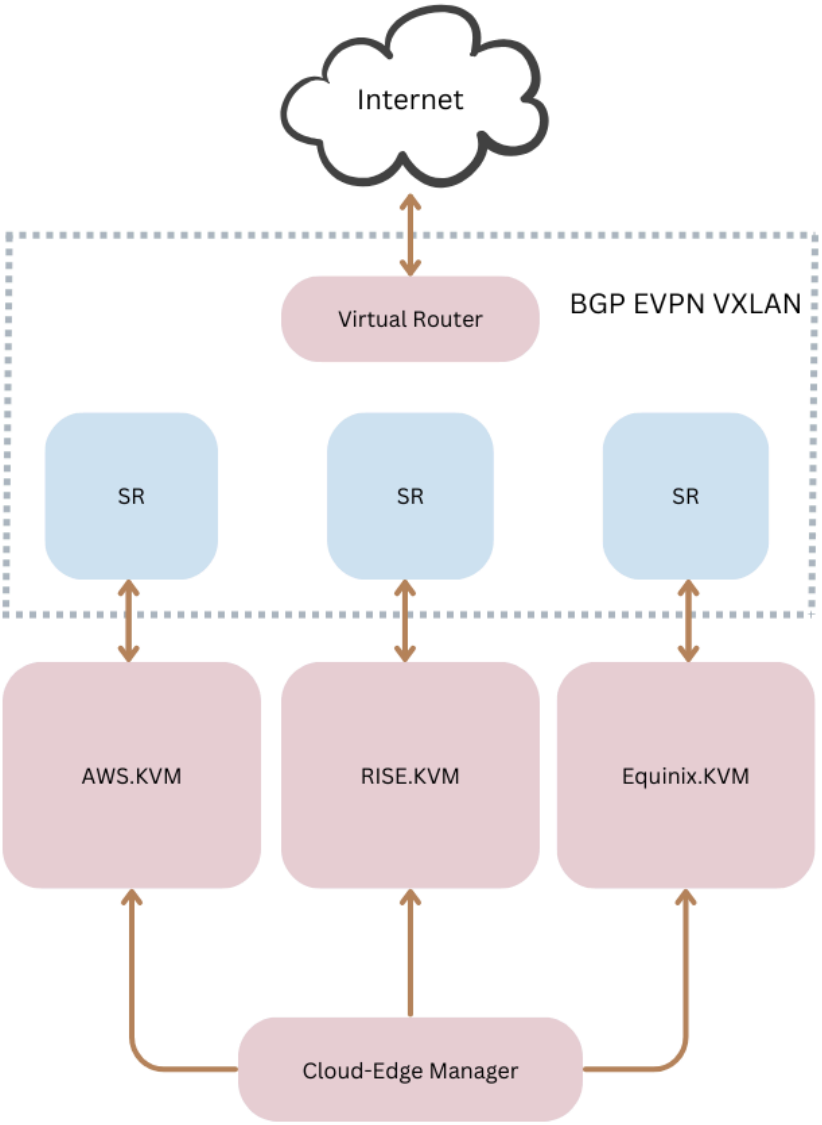

Figure 1.1: Cross-site Migration Network setup for OpenNebula clusters

When creating the network, each KVM node must be set up as an FRR routing daemon. The information coming from these daemons will be spread to the rest of the hypervisors using route reflectors. The KVM nodes need to reach the route reflector and vice-versa. Once a VM is deployed, it is connected to a L2 network encapsulated over the Internet. When migrating the VM to another host, the VM will remain in the same network. If the different KVM hosts do not share the same network interface names, the bridge that represents the VXLAN needs to be pre-created on each host with its PHYDEV. This particular configuration of the Cloud/Edge manager is applied by OpsForge in the automatically deployed Edge Clusters.

1.2 [SR4.4] Metrics, Monitoring, Auditing

Description

This cycle we focused on analysing the energy metrics mechanism based on Scaphandre that has been developed in the two first cycles, to explore the options available to extract energy consumption from specific software processes running on the Serverless Runtime. This information is key to provide the AI-Enabled Orchestrator with additional context on the consumption of different delegated functions coming from the Device Clients, in order to optimise the placement of Serverless Runtimes. After this analysis, we have settled on the following approach:

- Under an energy consumption perspective, every Serverless Runtime instantiated by the Cloud-Edge Manager can be considered a process. Scaphandre can compute the energy consumption for every process (i.e. every VM) as well as the host aggregated energy usage. In order to compute every Serverless Runtime internal process energy usage, a small agent can be developed to run on every VM, that computes the amount of CPU used by every process during a customizable amount of time and ponderate it with the overall Serverless Runtime energy consumption. In this way we may not only know the energy of every process dedicated to run offloaded functions, but also know if there is another system process or daemon that may be using too many resources on the Serverless Runtime.

In the next cycle we plan to develop this agent to extract per function energy consumption metric, and make it available to the AI-Enabled Orchestrator.

1.3 [SR4.5] Authentication and Authorization

1.3.1 Biscuit integration

The Biscuit auth driver for OpenNebula has been developed to allow the use of Biscuit tokens as a replacement for a user password when issuing API Calls.

Authentication in OpenNebula

In the OpenNebula database, there are two values saved for each user, they are **username** and **password**. When the driver used for authentication is of type ‘core’ (authenticated without an external auth driver), the password value holds the SHA256 hash of the user’s password. In case we are using another authentication method, this password field can contain any other information we can use to recognize a user.

In this Biscuit driver, the password value is not a password per se, but a public key is used to verify the token origin. Each user can generate tokens with their own private key, and register their public key as their password when creating the user. This public key will authenticate the tokens signed by the private key. This flow is depicted in Figure 1.2.

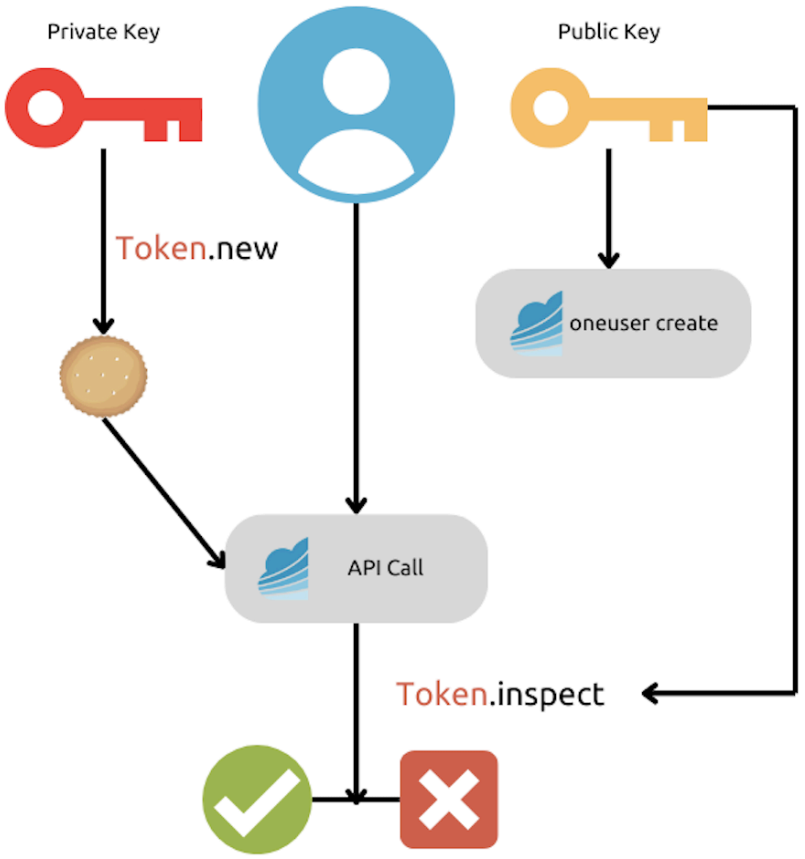

Figure 1.2.: Authentication Flow in OpenNebula Biscuit Integration

Deployment Details

This integration has not been contributed to the upstream OpenNebula repository, although there are plans to add it as it has been considered of interest for the general OpenNebula community.

Meanwhile it is available as an OpenNebula extension in the COGNIT Github organisation. It requires installing the Biscuit CLI on the Cloud/Edge Front-end, which can be achieved by simply downloading and making it available as introduced in Figure 1.3.

```
Unset
wget
https://github.com/biscuit-auth/biscuit-cli/releases/download/0.4.2/biscuit-
0.4.2-x86_64-unknown-linux-musl.tar.gz
tar -xf biscuit-0.4.2-x86_64-unknown-linux-musl.tar.gz
mv biscuit-0.4.2-x86_64-unknown-linux-musl/biscuit /usr/bin/
```

Figure 1.3: Installing Biscuit CLI

Afterwards the Cloud/Edge Frontend needs to have the relevant integration files available. This is achieved by simply copying the 'authentication' driver into the OpenNebula auth folder, in `/var/lib/one/remotes/auth`. To be able to use the new driver we need to add its name to the list of enabled drivers in the OpenNebula main daemon configuration file, a relevant excerpt is shown in Figure 1.4, and restarting all the OpenNebula services.

```
Unset
AUTH_MAD = [
    executable = "one_auth_mad",
    authn = "ssh,x509,ldap,server_cipher,server_x509,length,biscuit"
]
```

Figure 1.4: Activating the Biscuit Authentication driver.

Usage Details

In this section we are going to show a usage example to better understand how Biscuit and the Cloud-Edge Manager interact with each other and how they are integrated. Figure 1.5 shows how to create a user using the biscuit driver, provided that OpenNebula is correctly configured.

```
Unset
$public_key=41e77e842e5c952a29233992dc8ebbedd2d83291a89bb0eec34457e723a69526
oneuser create dann1 $public_key --driver biscuit
root@ubuntu2204-92:~# oneuser list
  ID NAME
ENAB GROUP    AUTH            VMS     MEMORY        CPU
   2 dann1
yes  users    biscuit     0 /   -      0M /   0.0 /   -
   1 serveradmin
yes  oneadmin server_c    0 /   -      0M /   0.0 /   -
   0 oneadmin
yes  oneadmin core              -          -          -
```

Figure 1.5: Create a user with Biscuit auth driver

`$public_key` is the public key used to verify the biscuit token sent by the user instead of the regular password auth. The private key must be kept safe by the user. The public key will act as the password for the user entry into the OpenNebula database. See here how to generate a keypair to sign tokens.

To use the driver to authenticate the user, a Biscuit token must be sent on the password argument when issuing an API Call. The token must contain the username in the authority block. Once the token is generated, it will act as a password when issuing API calls, as seen on Figure 1.6.

```
Unset
oneuser list --user dann1 --password
En4KFAoFZGFubjEYAyIJCgcIChIDGIAIEiQIABIgv37UGwg8lSQNVlEg3IN7JbJjhlgQh9eRU_fe
DPmmFwUaQIsybrP_UtZuvv0joaHvesZGNnZ5vFoxMCc18gtNB5FfrKDeNf3KBnSvs1Egd3CUegc5
BNhxkpli4vF5d4bsCQ4iIgogkIekfE40VgomPT7RZhuZCT09DHPukbrDQeHGhGlkvbk=
  ID NAME
ENAB GROUP    AUTH            VMS     MEMORY        CPU
   2 dann1
yes  users    biscuit     0 /   -      0M /   0.0 /   -
```

Figure 1.6. Using Biscuit token as Cloud/Edge Manager password

2. AI-Enabled Orchestrator

AI-Enabled Orchestrator (AI-O) is responsible for smart management of distributed cloud-edge resources according to the dynamic workloads, infrastructure policies, and energy usage policies. You can refer to Deliverable D4.2 for detailed description about the AI-Enabled Orchestrator. This development cycle makes progress on the model developments and AIOps pipeline for training, validation and model repository. However, another progress is being carried out on energy-aware continuum systems modelling and its validation with simulated environments.

2.1 [SR5.1] Building learning models

In this development cycle, AI-O is implemented and integrated with the AIOps pipeline, in particular developing AI/ML models for resource prediction.

2.1.1 AI/ML Models

Deep learning (DL) models [13] have shown superior performance over classical machine learning models, particularly in prediction tasks in time series data, though they are often more data-hungry and computationally expensive. For instance, Long Short-Term Memory (LSTM) [1, 9] and Gated Recurrent Unit (GRU) [12] networks belong to the category of recurrent neural networks (RNNs), which are specifically designed to capture complex temporal patterns and regularities in sequential data. These mechanisms are extensively employed in Transformer models, which can enhance the analysis of time series data for various downstream tasks, including prediction.

As part of AI/ML model development, the AI-Orchestrator (AI-O) will maintain a model repository tailored to these diverse downstream tasks. Models are selected at runtime according to the task at hand, allowing the orchestrator to adapt to dynamic needs. The AI-O can leverage four deep learning models for prediction tasks: 1) Long Short Term Memory (LSTM), 2) Feed-Forward Neural Network (FFNN), 3) Temporal Convolutional Network (TCN), and 4) Gated Recurrent Unit (GRU).

1) LSTM

Long Short-Term Memory (LSTM) [1, 9] networks are designed to capture dependencies in sequential data over long periods of time, making them ideal for time series forecasting in cloud and edge environments. LSTMs address the vanishing gradient problem often faced by traditional RNNs, enabling them to retain important information over extended sequences. By learning from historical workload data, LSTM models can effectively predict future workloads, which aids in optimizing resource allocation and balancing computational loads across cloud-edge infrastructure. This capability is essential for tasks like workload prediction, proactive migration, and resource optimization. An example LSTM block and LSTM chain are illustrated in Figure 2.1 and 2.2, respectively.

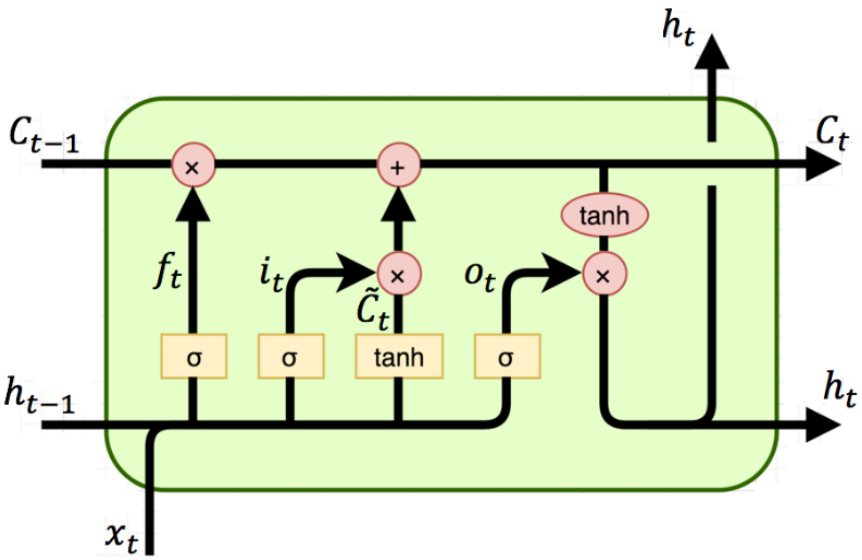

Figure 2.1: Architecture of a LSTM block.

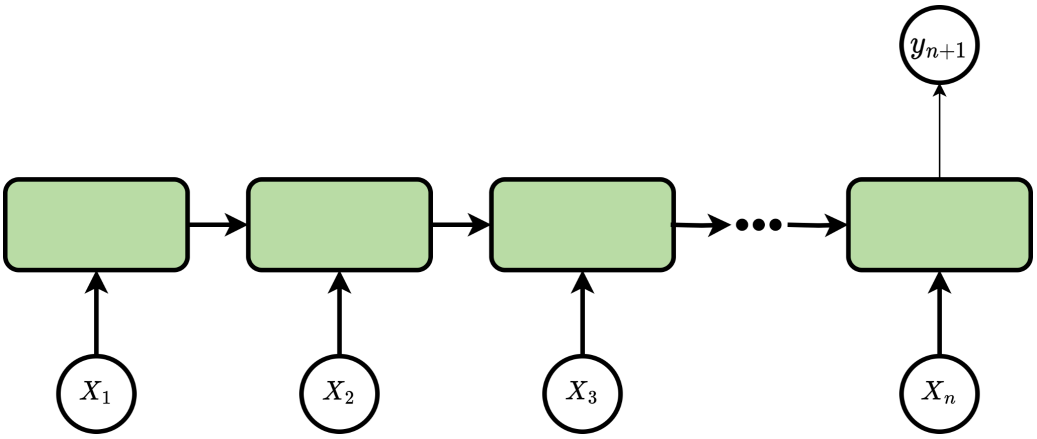

Figure 2.2: A LSTM chain consisting of several LSTM blocks.

LSTMs can also model non-linear dependencies between different metrics, such as CPU usage, memory consumption, and network traffic, making them highly suited for workload characterization. Furthermore, their ability to handle irregular intervals and multi-step forecasting makes LSTM networks a powerful tool for predictive maintenance and energy-aware placement in distributed cloud-edge continuum. Despite being computationally expensive, their high accuracy and ability to generalize well to unseen data justify their usage in performance-critical environments like AI-Enabled orchestrators.

2) FFNN

Feed-Forward Neural Networks (FFNNs) [10] are one of the simplest and most widely used types of neural networks. Unlike recurrent networks, FFNNs do not have any temporal dependencies, meaning they process input data in a straightforward manner from input to output layers. This makes them suitable for

tasks where dependencies between consecutive time steps are minimal or can be summarized in aggregated features. In the context of cloud-edge workload management, FFNNs can efficiently perform workload prediction and estimate resource utilization, where the focus is on capturing non-sequential patterns in the data. An example FFNN is presented in Figure 2.3.

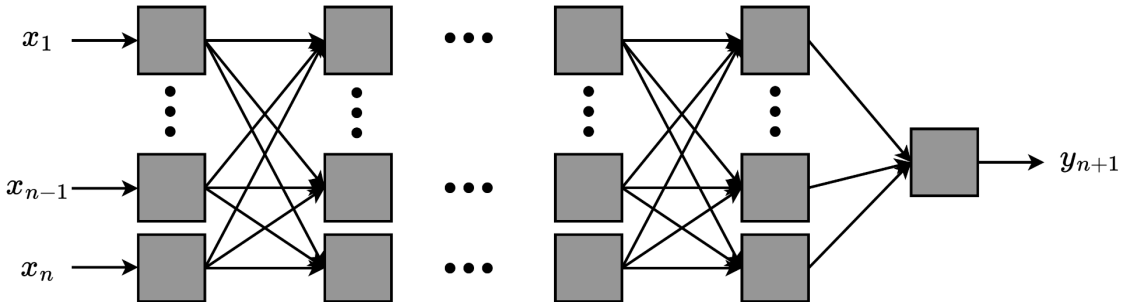

Figure 2.3: An example FFNN.

FFNNs excel when processing large datasets due to their parallel architecture, making them faster and computationally cheaper compared to more complex models like LSTMs and Temporal Convolutional Networks (TCNs). They are also easier to train and fine-tune, which can be advantageous in real-time decision-making within the orchestrator.

For example, FFNNs can be used to rapidly predict workloads based on resource needs in resource utilization patterns. While FFNNs may not capture temporal dependencies, they remain a valuable option for quick decision-making tasks and scenarios where simplicity and speed are prioritized over deep temporal analysis.

3) GRU

Gated Recurrent Unit (GRU) [12] networks are a type of recurrent neural network (RNN) that have been developed as a simpler alternative to Long Short-Term Memory (LSTM) networks. GRUs share a similar architecture but have fewer parameters and a more streamlined structure, making them computationally less expensive while still capable of capturing temporal dependencies in sequential data. Like LSTMs, GRUs are well-suited for time series forecasting, especially when resource efficiency is a priority, such as in edge computing environments.

The key difference between GRU and LSTM networks lies in the gating mechanism. GRUs use a combined update gate and do not maintain a separate memory cell, making them faster to train while still retaining sufficient long-term dependencies in the data. This makes GRU a promising model for real-time workload prediction, where balancing performance and computational overhead is critical. GRU's ability to process sequential data with fewer computations makes it particularly useful in resource-constrained environments like the cloud-edge continuum. Despite being simpler, GRUs can still capture non-linear relationships between metrics such as CPU usage, memory consumption, and network traffic, making them effective in

tasks like workload prediction, dynamic resource allocation, and latency-sensitive applications. An example GRU block is illustrated in Figure 2.4.

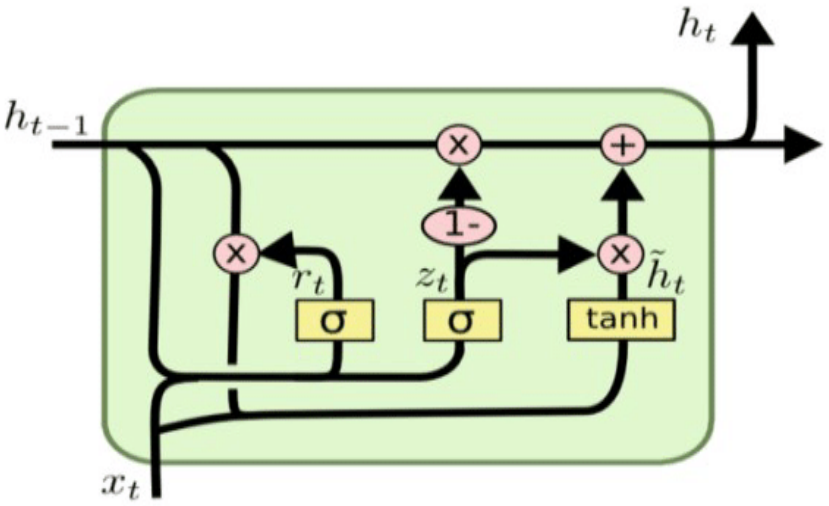

Figure 2.4: Architecture of a GRU block.

4) TCN

Temporal Convolutional Networks (TCNs) [11] provide a novel approach to time series prediction by leveraging convolutional layers instead of recurrent neural network-based architectures. Unlike LSTMs, TCNs use 1D dilated causal convolutions to capture temporal dependencies, which allows them to process long sequences of data more efficiently. This makes TCNs particularly effective in modelling complex patterns over long time horizons while maintaining a simpler, more parallelizable architecture. In the AI-Enabled Orchestrator, TCNs can be applied to tasks such as long-term workload forecasting and trend analysis, where both local and global temporal features are essential. An example TCN is presented in Figure 2.5.

TCNs are designed to maintain the sequence order and prevent data leakage from future time steps, ensuring that predictions are based solely on past data. Additionally, their residual connections improve training stability, making TCNs a competitive alternative to LSTMs, especially in environments with high-dimensional data like cloud-edge infrastructures. TCNs can outperform traditional RNN-based models in terms of both speed and accuracy, particularly in tasks like resource demand prediction, capacity planning, and proactive scaling. Their ability to capture temporal dependencies across multiple time scales makes TCNs a versatile and powerful choice for orchestrating dynamic cloud and edge resources.

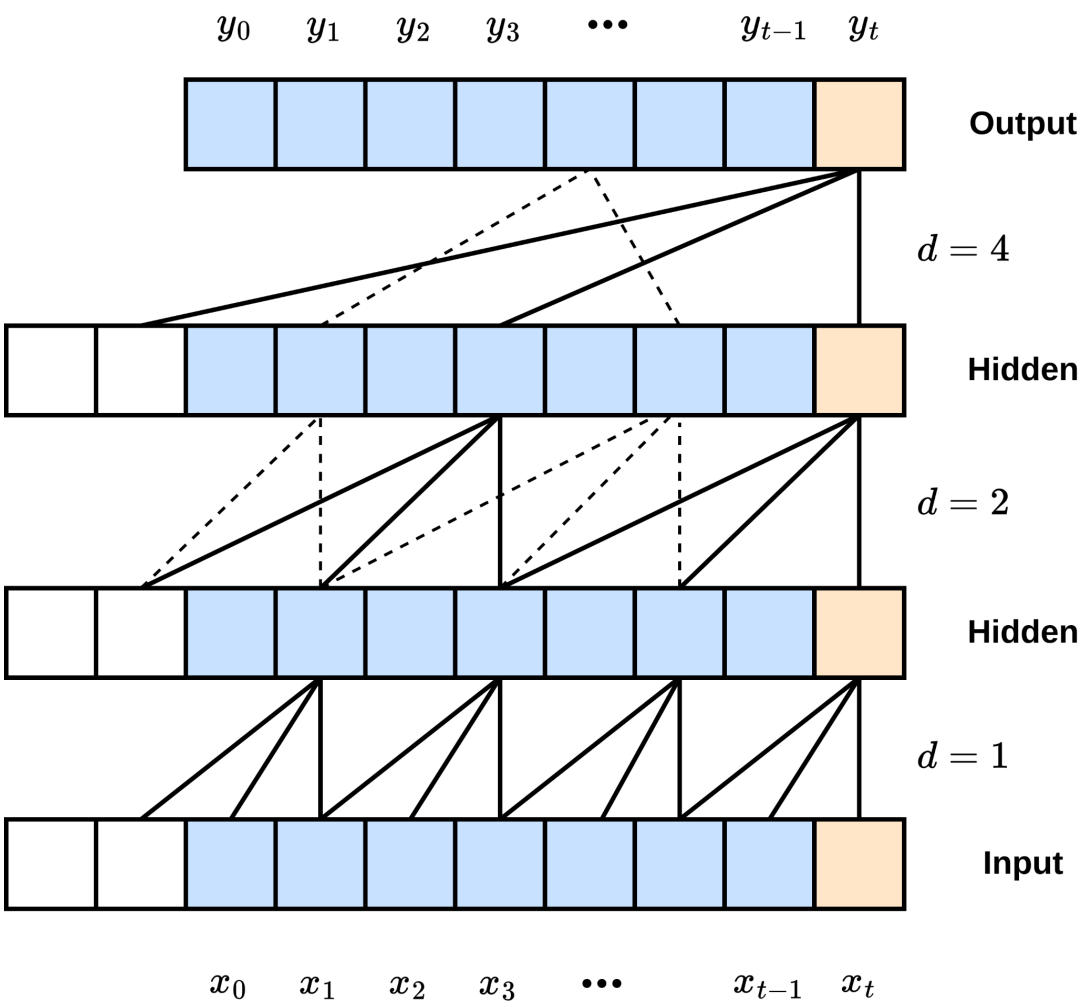

Figure 2.5: An example TCN.

2.1.2 Data

These models have been developed, tuned, validated and integrated with the AIOps pipeline in COGNIT using the GWA-t-12 Bitbrains public dataset, which provides comprehensive performance metrics from virtual machines in a distributed datacenter. Also, they are validated with testbed and emulator data. For more details on the additional data sources and emulator setup, which was used as well as will be used in future evaluations, please refer to Deliverable D4.2.

2.1.3 Results and Analyses

In this part, we describe our approach to implementing and training various neural network architectures for the resource consumption prediction tasks. Our objective was to compare the performance of four different models—LSTM, GRU, FFNN, and TCN—on predicting resource consumption metrics, including CPU usage, memory usage, disk write speed, and network activity.

a) Data Preparation and Preprocessing

- **Data Loading**: We began by loading the multivariate time series data for the aforementioned metrics.
- **Data Scaling**: The data was scaled to normalise the feature values and ensure consistent model training.
- **Sequence Creation**: Using a sequence length of 99 and a prediction length of 1, we created input-output pairs suitable for training time series models.
- **Training and Testing Split**: The data was split into training and testing sets with an 80:20 ratio.

b) Model Architecture

We selected four neural network architectures suited for time series forecasting:

- **LSTM**: The Long Short-Term Memory model (LSTM) is configured with 2 layers, an input size of 4, a hidden size of 64, and an output size of 4. This structure allows it to retain long-term dependencies in the data, making it effective for sequential data.
- **GRU**: The Gated Recurrent Unit (GRU) model has a similar configuration to the LSTM, with an input size of 4, hidden size of 64, and output size of 4. GRUs are more computationally efficient than LSTMs while still capturing essential time dependencies.
- **FFNN**: The Feedforward Neural Network (FFNN) model uses a fully connected architecture. Its input size was adjusted to account for sequence length, enabling it to process flattened time series data. The hidden layer dimension is set to 64.
- **TCN**: The Temporal Convolutional Network (TCN) model consists of three convolutional layers, with 64 filters in each layer. The kernel size is set to 2, and the model uses dilation to capture long-term dependencies without recurrent connections.

c) Training and Hyperparameters

All models were trained using the following hyperparameters:

- **Batch Size**: 64
- **Learning Rate**: 0.001
- **Epochs**: 1 (for demonstration purposes; could be increased for more robust performance analysis).
- **Loss Function**: Mean Squared Error (MSE).
- **Optimizer**: Adam, selected for its adaptive learning rate, which helps in optimising the model efficiently.

d) Evaluation Process

- **Prediction Performance**: We evaluated each model on the test set using Root Mean Squared Error (RMSE) for each resource consumption metric. The RMSE

comparison of all models is provided in Table 2.1. Additionally, we calculated the average prediction time per sample to assess model efficiency which is presented in Table 2.2.

- **Visualisation**: For each model, we plotted the predicted vs. actual values for a random sample of 100 test instances to visually inspect the model's performance across metrics.

This experimental setup enabled a comprehensive comparison of each model's performance on resource consumption metrics prediction, leading to detailed insights starting with our analysis of the following models:

1) LSTM

In addition, comparison of the model predictions compared to the ground truth is provided in Figure 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12 and 2.13 by considering 100 random samples for the illustration. The Long Short-Term Memory (LSTM) model demonstrates strong performance in predicting several key workload metrics. Specifically, LSTM outperforms the Feed-Forward Neural Network (FFNN) in predicting Memory Usage and Disk Write with RMSE values of 0.0018 and 0.0024, respectively, showcasing its ability to capture complex temporal dependencies in time-series data. The green lines in Figure 2.6 and 2.7 represent the ground truth, while the purple dashed lines indicate the predicted values. The closeness of these two lines for Memory and CPU usage highlights the LSTM's accuracy in following the actual trends over time. This suggests that LSTM is particularly effective in scenarios where long-term dependencies or sequential patterns are crucial, such as resource allocation and proactive scaling in cloud-edge environments.

On the other hand, LSTM’s prediction for CPU Usage and Network Received metrics, while still competitive, shows a slightly higher RMSE compared to FFNN in the case of CPU Usage (RMSE 0.0392). However, for Network Received throughput, both models performed similarly, with an RMSE of 0.0004, indicating that LSTM is capable of maintaining accuracy across different types of workload metrics. The visualizations indicate that while the LSTM’s predictions occasionally deviate from the actual values, it still manages to capture the general patterns of CPU spikes and drops. This makes it a strong candidate for tasks where understanding periodic fluctuations in workload is critical for optimization.

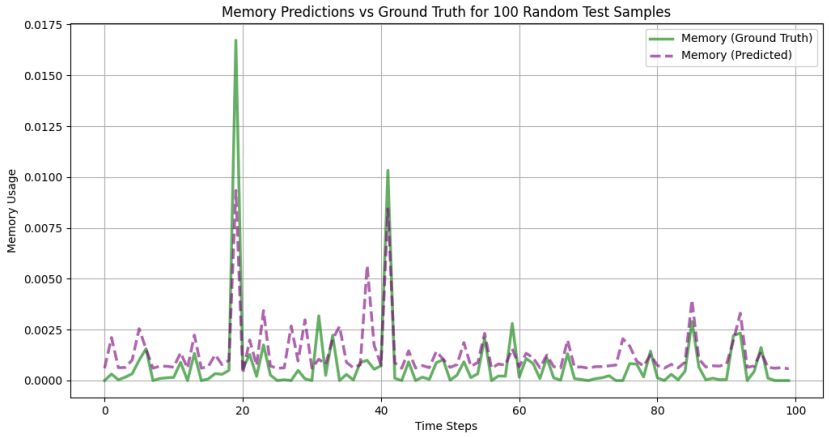

Figure 2.6: Illustration of memory usage vs the prediction by LSTM.

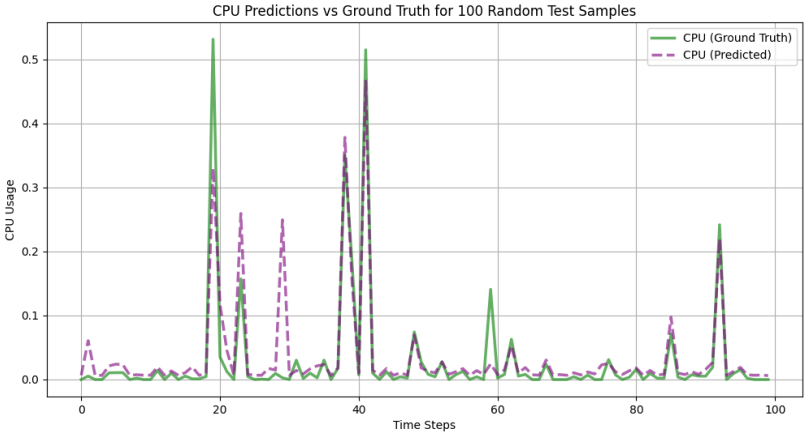

Figure 2.7: Illustration of CPU usage vs the prediction by LSTM.

2) FFNN

The Feed-Forward Neural Network (FFNN), despite being a simpler model, demonstrates notable performance, particularly in predicting CPU Usage, where it outperforms LSTM with a lower RMSE of 0.0245. This indicates that FFNN, which does not rely on capturing sequential dependencies, can be highly effective in predicting non-temporal metrics or those with fewer long-term dependencies. Figure 2.8 and 2.9 corresponding to Memory and CPU Usage show a good overlap between the FFNN predictions (purple dashed line) and the actual values (green line), suggesting that FFNN is capable of capturing the immediate fluctuations and short-term variations in the data, making it suitable for real-time decision-making tasks in resource management.

However, FFNN struggles more with predicting Memory Usage and Disk Write, with RMSE values of 0.0021 and 0.0034, respectively, higher than those of the LSTM. This suggests that the FFNN may have difficulty capturing more complex temporal relationships that are crucial for metrics with a more sequential nature. In contrast,

for Network Received, the FFNN's performance is very similar to LSTM, with only a marginal difference in RMSE (0.0005 vs. 0.0004), implying that both models are equally competent in predicting network-related metrics, where immediate input-output relationships might dominate over temporal dependencies.

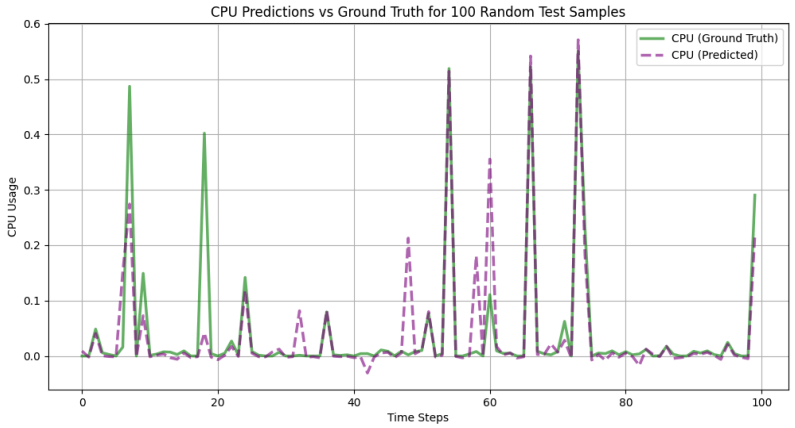

Figure 2.8: Illustration of memory usage vs the prediction by FFNN.

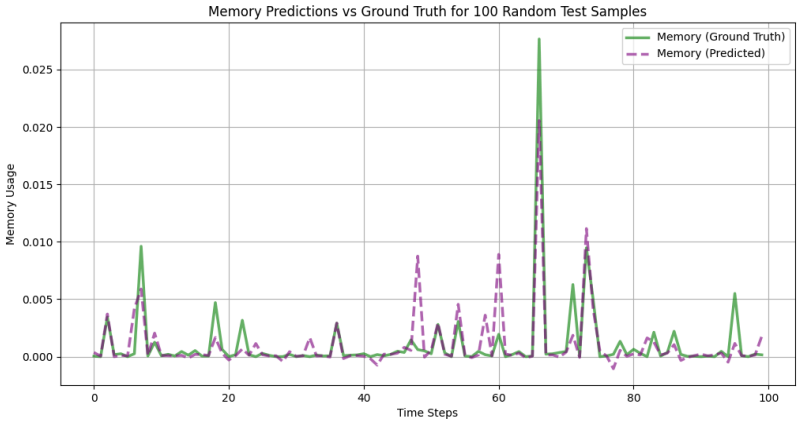

Figure 2.9: Illustration of CPU usage vs the prediction by FFNN.

3) GRU

The RMSE comparison of the GRU model is provided in Table 2.1, alongside the other models (LSTM and FFNN). The predictions generated by the GRU model are compared to the ground truth in Figures 2.10 and 2.11, using 100 random samples for illustration. Overall, GRU demonstrated competitive performance across several workload metrics, although it did not outperform the LSTM and FFNN in most cases. Specifically, GRU produced an RMSE of 0.0623 for CPU Usage, which is higher than both LSTM and FFNN, indicating that it struggled more with capturing long-term dependencies in CPU usage data. The ground truth (green lines) and predicted values (purple dashed lines) for CPU usage, shown in Figure 2.10, reveal that while GRU follows the general trend, it has difficulty aligning with CPU spikes and drops as accurately as the LSTM and FFNN.

For Memory Usage, GRU yielded an RMSE of 0.0027, which is slightly higher than LSTM's 0.0018 and FFNN's 0.0021. This suggests that GRU can still capture short-term dependencies effectively for memory predictions, though LSTM outperforms it in capturing the more complex temporal dynamics. Similarly, for Disk Write throughput, GRU's RMSE was 0.0050, higher than both LSTM and FFNN, indicating that it may not be the best model for storage-related predictions.

Lastly, for Network Received throughput, GRU achieved an RMSE of 0.0006, which is close to LSTM and FFNN but still slightly higher. While the overall trend is captured, results indicate that GRU occasionally diverges from the actual values.

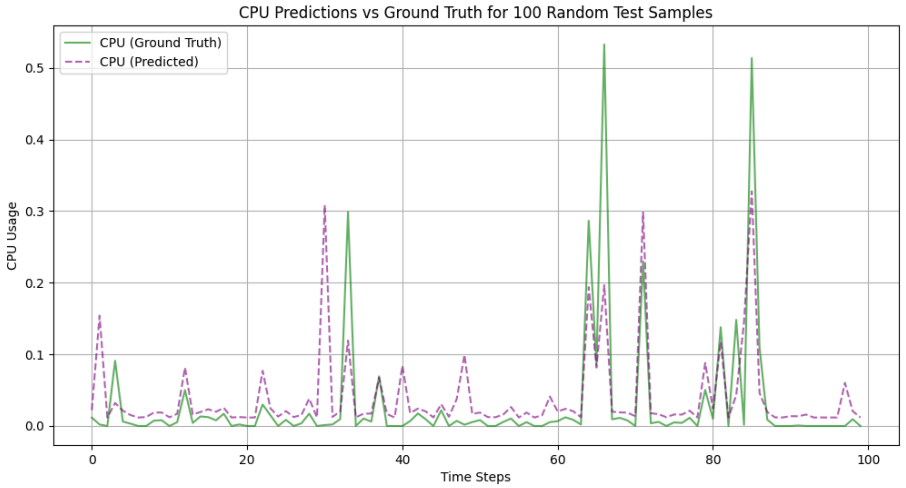

Figure 2.10: Illustration of CPU usage vs the prediction by GRU.

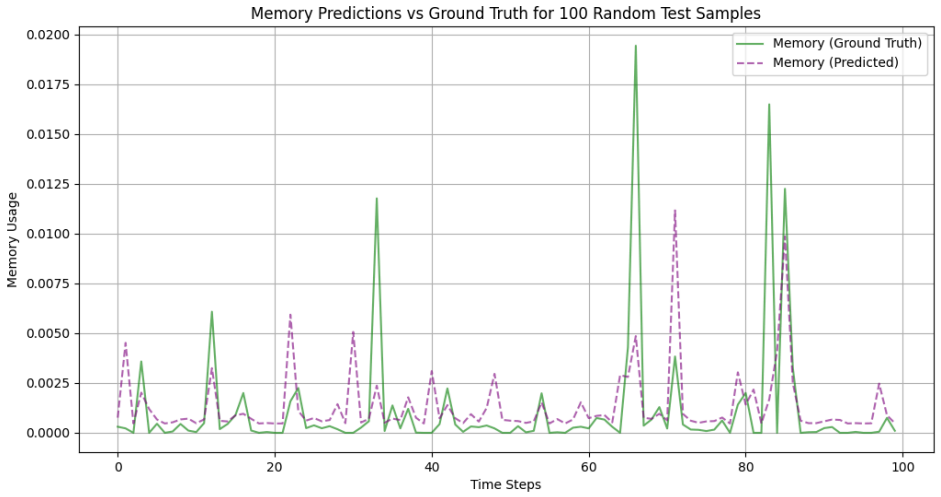

Figure 2.11: Illustration of memory usage vs the prediction by GRU.

4) TCN

The RMSE comparison of the TCN model is provided in Table 2.1, alongside the other models (LSTM, FFNN, and GRU). The predictions generated by the TCN model are compared to the ground truth in Figures 2.12 and 2.13, using 100 random samples for illustration. TCN demonstrated competitive performance across several workload metrics, outperforming most models in some cases, particularly in Memory Usage predictions. Specifically, TCN produced an RMSE of 0.0483 for CPU Usage, which is lower than GRU's 0.0623 but higher than both LSTM's 0.0392 and FFNN's 0.0245. As shown in Figure 2.12, the ground truth (green lines) and predicted values (purple dashed lines) for CPU usage reveal that TCN aligns reasonably well with the general trend but encounters slight difficulties in tracking sharp CPU spikes, similar to GRU.

For Memory Usage, TCN excelled with an RMSE of 0.0012, which is the lowest among all models, indicating that it can effectively capture short-term dependencies and temporal dynamics in memory data. This is clearly illustrated in Figure 2.13, where the predicted memory usage closely follows the ground truth across random samples.

In terms of Disk Write throughput, TCN produced an RMSE of 0.0046, which is lower than GRU but higher than LSTM and FFNN. While the model captures the overall trend of disk usage, it may face challenges in predicting rapid fluctuations in storage-related metrics.

Lastly, for Network Received throughput, TCN achieved an RMSE of 0.0007, which is slightly higher than all other models but still comparable. Though the differences are minimal, TCN occasionally diverges from the true values, especially when network traffic undergoes abrupt changes. Overall, TCN provides strong performance and offers a reliable option for temporal data prediction, particularly for memory and CPU usage.

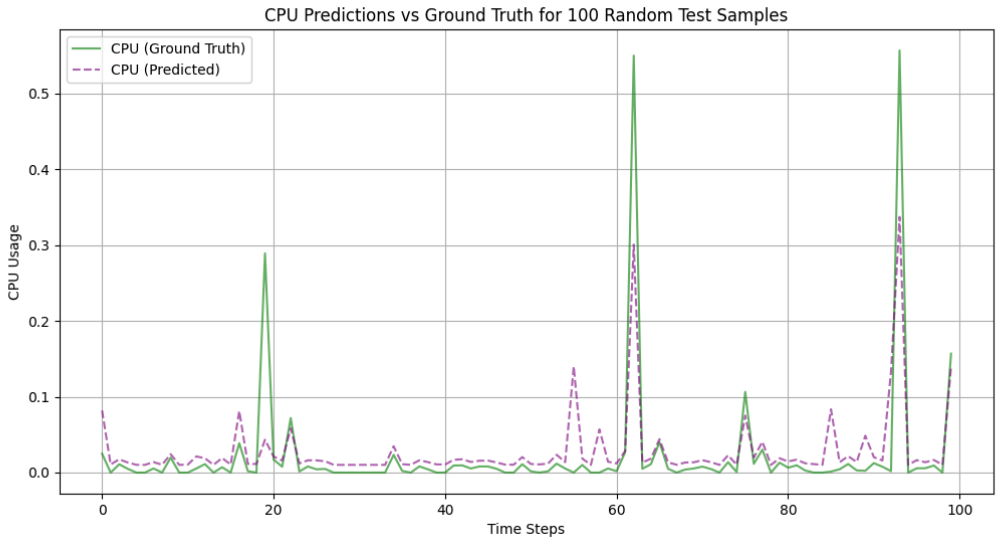

Figure 2.12: Illustration of CPU usage vs the prediction by TCN.

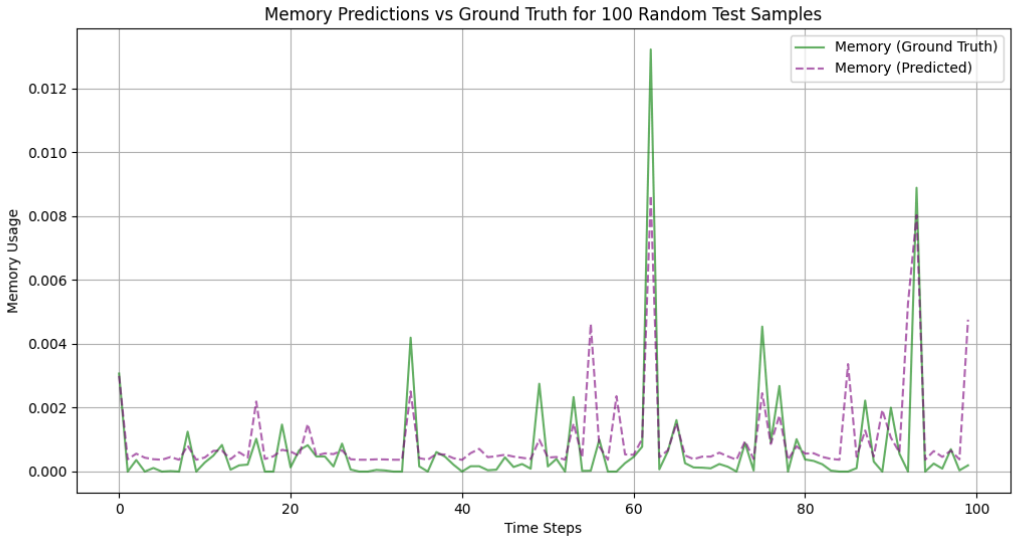

Figure 2.13: Illustration of Memory usage vs the prediction by TCN.

Metric	LSTM RMSE	FFNN RMSE	GRU RMSE	TCN RMSE
CPU Usage [%]	0.0392	0.0245	0.0623	0.0483
Memory Usage [KB]	0.0018	0.0021	0.0027	0.0012
Disk Write [KB/s]	0.0024	0.0034	0.0050	0.0046
Network Received [KB/s]	0.0004	0.0005	0.0006	0.0007

Table 2.1: Comparison of LSTM, FFNN, GRU, and TCN over RMSE for different system metrics.

Time Analysis

The time required for making predictions is a critical factor when selecting models for real-time or near-real-time applications in cloud and edge environments. The Prediction Time in milliseconds for LSTM, FFNN, and GRU models, shown in Table 2.2, provides insight into their computational efficiency and scalability under various operational constraints.

- FFNN stands out as the fastest model with a prediction time of 0.14 milliseconds. This is expected, as the FFNN does not involve recurrent connections or sequential dependencies, making it highly efficient for making predictions. Its simpler architecture allows it to process inputs and generate outputs with minimal computational overhead, making it an excellent choice for applications where prediction speed is crucial and the workload patterns are non-temporal or short-term.

- LSTM and GRU, on the other hand, show significantly higher prediction times of 7.39 milliseconds and 7.54 milliseconds, respectively. Both models are designed to capture temporal dependencies in the data, which comes at the cost of increased complexity. While LSTM's prediction time is marginally lower than GRU, the difference is negligible in most real-time systems. These models are ideal when it is critical to capture long-term dependencies in workload patterns, even though they are slower compared to FFNN. The added complexity in these models makes them suitable for scenarios where accuracy in temporal prediction outweighs the need for ultra-low latency in predictions.
- TCN strikes a balance between recurrent models like LSTM/GRU and feedforward models like FFNN. With a prediction time of 1.38 milliseconds, it is significantly faster than both LSTM and GRU, while maintaining its capability to capture temporal dependencies. TCN's architecture allows for parallel processing across temporal layers, resulting in a more efficient prediction process compared to recurrent networks. This makes TCN an attractive choice for tasks that require a compromise between speed and the need to capture sequential data, offering better scalability in time-sensitive applications than LSTM or GRU without sacrificing much predictive accuracy.

Model	LSTM	FFNN	GRU	TCN
Prediction Time (msec)	7.39	0.14	7.54	1.38

Table 2.2: Average prediction time per instance for each model.

2.2 [SR5.2] Smart Management of Cloud-Edge Resources

Managing cloud-edge resources is key in AI-Enabled Orchestrators across the continuum by employing the outcomes of AI/ML models such as prediction of workload with respect to dynamic resource consumption (CPU, memory, network), energy consumption. The systems are modelled in two ways, one is straightforward development of algorithms for resource optimization and integration with COGNIT testbed, and on the other hand, in order to understand the scalability of the system across the continuum, modelling and simulations are explored and reported in subsection 2.2.2.

2.2.1 Cloud-Edge resource optimization algorithms

Optimising cloud-edge resources is crucial to achieve optimal resource management across the continuum by adapting the dynamic workload and infrastructure policies. In this development cycle, interference and energy aware resource optimization is explored.

a) Metrics

Metrics for application and infrastructure are collected by Cloud-Edge manager, which are utilised to develop optimization algorithms and taking into consideration some constraints such as latency, green energy host.

b) Algorithms

b1. Energy-interference optimization algorithm:

Optimising energy and interference is key for managing cloud-edge resources while ensuring execution, migration, scaling of serverless runtimes. Here, the first objective is to maximise the green energy host usage and minimise the workload interference according to the dynamic load, which is indeed conflicting objectives to meet. This section details an advanced algorithm that employs predictive models, multi-objective optimization via NSGA-II, and energy-aware scheduling mechanisms to achieve these goals in heterogeneous continuum environments.

b1.1 Predict interference with AI/ML models

In these continuum scenarios, application performance can be significantly impacted by resource contention, where virtual machines (VMs) or serverless runtimes (SR) share physical resources like CPU, memory, and network bandwidth. This resource sharing leads to interference, reducing performance predictability and complicating resource allocation.

To predict and mitigate interference, machine learning (ML) models are trained using historical data on workloads and resource usage, which enables more proactive scheduling decisions.

Define the following variables for interference prediction:
$W_i(t)$: Resource usage vector at time *t* for SR *i*, encompassing CPU, memory, and I/O demands.
$QoS_i(t)$: Quality of Service level for SR *i*, measured by latency and throughput requirements.
$I(t)$: Predicted interference metric at time *t*, which represents the degree of resource contention among co-hosted VMs.

Using this setup, the ML model's output $\widehat{I}(t)$ (predicted interference) is computed as:

$$\widehat{I}(t) = f(W_i(t),\ QoS_i(t)) \qquad for\ all\ i,$$

where *f* is the model trained to capture the relationship between workload characteristics and resource interference. Real-time predictions of $\widehat{I}(t)$ are essential for minimising latency deviations and for adjusting VM allocations to avoid performance degradation due to interference [16].

b1.2 NSGA-II: Minimise Interference and Maximise Green Energy Usage

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is employed to balance two key objectives: minimising interference and maximising the use of green energy resources. NSGA-II operates by iteratively refining a set of candidate solutions through a process of selection, crossover, and mutation to achieve Pareto-optimal solutions [15].

1) Multi-Objective Optimization with NSGA-II

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is a multi-objective evolutionary algorithm used to address conflicting goals in complex optimization problems. Here, NSGA-II helps balance interference reduction and energy conservation across cloud-edge clusters by optimising two primary objectives:

1. Objective 1 - Minimise Interference (f_1):

$$f_1 = min \sum_{i=1}^{n} I(W_i(t),\ QoS_i(t))$$

where $I(W_i,\ QoS_i)$ quantifies the interference experienced by SR i under its current resource and QoS requirements.

2. Objective 2 - Maximise Green Energy Usage (f_2):

$$f_2 = max \sum_{j=1}^{m} G_j \cdot y_j$$

where G_j represents the availability of green energy (e.g., solar, wind) at datacenter j, and y_j is a binary variable indicating whether green energy is selected for datacenter j.

Each individual in the NSGA-II population represents a possible configuration of SR placements and green energy usage. NSGA-II evolves the population by iteratively selecting, mutating, and recombining solutions based on **non-domination sorting** and **crowding distance** to explore the Pareto front, where the set of optimal trade-offs between minimising interference and maximising green energy usage is identified [15, 16].

Variables and Constraints

To formalise the NSGA-II optimization problem, we introduce the following variables and constraints:

- x_{ij}: Binary decision variable, where $x_{ij} = 1$, if SR *i* is allocated to server *j*; otherwise, $x_{ij} = 0$
- $E_j(t)$: Green energy consumed by server *j* at time *t*, which must align with its green energy budget.
- D_K: Deadline for serverless function *k*, dictating when execution must complete to meet QoS standards.

Constraints:

1. **Resource Capacity:** The total resource demand of SRs on each server cannot exceed its available capacity:

$$\sum_{i=1}^{n} W_i(t) \cdot x_{ij} \leq R_j \quad \text{For all servers } j,$$

where R_j is the resource capacity of server *j*.

2. **QoS Compliance**: Each SR's performance should meet its QoS requirements, given by:
$$QoS_i(t) \geq Qos\ threshold$$
For SR *i* to avoid service level agreement (SLA) penalties.
3. **Green Energy Consumption Limit**: The total energy consumption $E_j(t)$ at each server should not exceed available green energy when possible:
$$E_j(t) \leq G_j \cdot y_j$$

2) Energy-Interference aware Placement of Serverless Runtimes

The scheduling of serverless runtimes involves dynamically allocating functions across cloud-edge resources based on current energy availability and interference levels. For each serverless runtime chain F_k with computational load L_k and deadline D_k, the following actions are taken:

A. **Function Implementation Selection:** Each function F_k may have multiple implementations, $F_{k_1}, F_{k_2}, \ldots, F_{k_n}$, each with distinct trade-offs between computational accuracy and green energy consumption. The scheduler prioritises implementations with green energy footprints when energy constraints are tight, using approximate computing techniques [17].

B. **Computational Offloading:** When an edge node's battery level is low or when green energy sources are depleted, the scheduler can offload computation to cloud nodes. Offloading reduces local energy consumption and ensures function execution deadlines D_k are met .

C. **Algorithmic Steps for the Energy-Interference Optimization:**
   a. **Input:** Serverless Runtimes, F, deadlines D, green energy budget G, interference predictions I.
   b. **Initialize:** Generate an initial set of solutions for SR placements and green energy usage.
   c. **Evaluate:** For each solution, calculate interference f_1 and green energy usage f_2.
   d. **Select and Evolve:** Use non-dominated sorting and crowding distance to retain Pareto-optimal solutions, applying crossover and mutation.
   e. **Output:** Deploy the best solution that optimally reduces interference and maximises green energy usage efficiency.

The plan is to evaluate and integrate the energy-interference optimization using COGNIT testbed, benchmark datasets, and emulators (that is ongoing development).

By integrating AI/ML for interference prediction, NSGA-II for multi-objective optimization, and energy-aware scheduling, this algorithm provides a robust solution to balance performance with sustainability in the Cloud-Edge continuum. This approach not only optimises green energy usage but also ensures service-level compliance, making it suitable for various energy-sensitive and QoS-critical applications in serverless computing.

b1.3 Evaluation strategies

- Prediction accuracy for interference within Edge Cluster when co-locating execution of Serverless Runtimes.
- Estimate waiting time of function execution - in absence and in presence of energy- interference optimization algorithm for Cloud-Edge resource management.

2.2.2 System Modelling and Simulations

a) Stable matching based modelling for energy-aware continuum systems

Continuum systems are dynamic, often massive in scale, and feature disparate infrastructure providers and platforms; this greatly increases the complexity of developing and managing applications. The Serverless paradigm shows the potential to greatly simplify the process of building Continuum applications - however, current scheduling mechanisms for Serverless Continuum platforms pay little attention to reducing the energy consumption and improving the sustainability of function execution.

This is a significant omission, made worse as computing nodes within a Continuum may be powered by renewable energy sources that are intermittent and unpredictable, making low-powered and bottleneck nodes unavailable. There is great opportunity to design a decentralised energy management scheme for scheduling Serverless functions that takes advantage of the different layers of the Continuum, such as IoT devices, edge nodes,on-premises clusters closer to the data sources, or directly on large Cloud infrastructures.

Through COGNIT, we formally model a green energy-aware Serverless workload scheduling problem for the multi-provider Cloud-Edge Continuum, published in [2]. In this work, we formally model the problem of decentralised energy management (utilising a distributed controller) that considers the availability of green energy nodes and the QoS requirements of Serverless functions as shown in Figure 2.14.

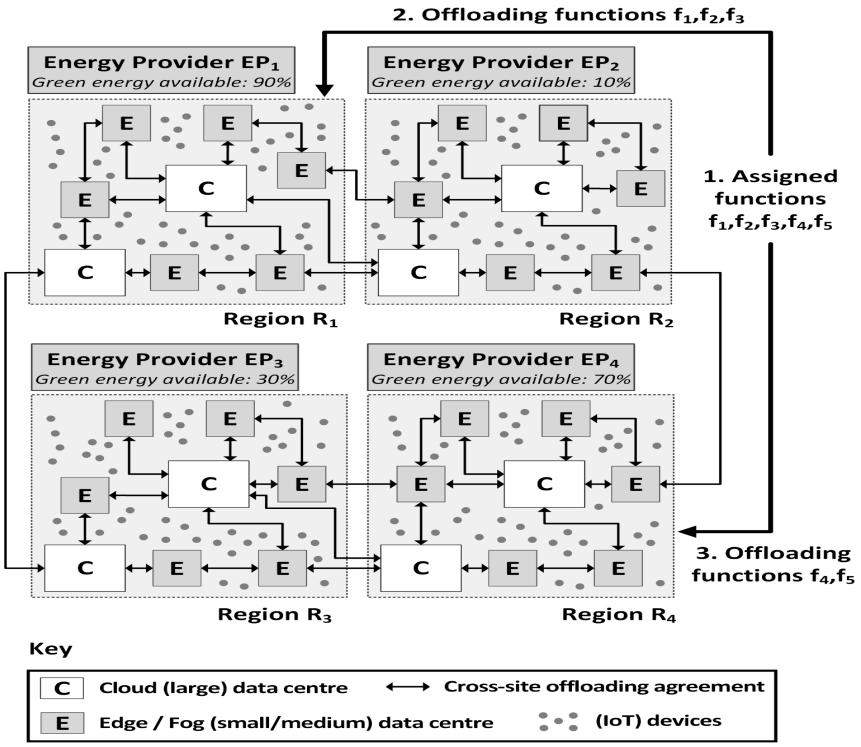

Figure 2.14: Scenario for scheduling functions in energy-aware Continuum systems

a1. Mathematical modelling

In this model, the scheduling functionalities are distributed and thus every node within a region is able to schedule the execution of incoming requests. This is especially crucial for requests generated at the edge. The request comprises several microservices, and the function executes tasks associated with each microservice. Each region possesses a local controller, which follows the IBM MAPE-K [3] (Monitoring, Analysis, Planning, Execution, and Knowledge) model. The MAPE-K serves as the reference control model for autonomic and distributed self-adaptive systems. It is designed to accurately capture the dynamics of the system and make effective decisions based on the available data and knowledge. Apart from the local controllers (LCs), there exists a global controller (GC) which provides information on all regions deployed in the continuum. Next, we formally model the stable matching for Continuum systems.

- Let $T = \{t | t \in [0, T], (t + 1) - t = \Delta t \; seconds\}$ be the set of consecutive time slots.
- Let $R = \{j \in [1, m]\}$ be the set of regions.
- Let $D = \{d_i^j | i \in [1, n], j \in [1, m]\}$be the set of n heterogeneous computing nodes located across m different regions.
- Let $G = \{g_{it}^j, | i \in [1, n], j \in [1, m], t \in T, g_{it}^j \geq 0\}$ be the energy input to node i.
- Let $E = \{e_{it}^j | i \in [1, n], j \in [1, m], t \in T, e_{it}^j \geq 0\}$ be the energy consumption of nodes.
- Let $S = \{s_{it}^j, | i \in [1, n], j \in [1, m], t \in T, \delta \geq s_{it}^j \geq 0\}$ be the available amount of SoC (State of Charge) on computing nodes.

- Let $Cap = \left\{ i \in [1,n], j \in [1,m], t \in T, cap_{it}^{j} \in [0,\gamma] \right\}$ be the overall resource capacity of nodes in MIPS (Million Instructions Per Second).
- Nodes can host functions if they satisfy the minimum energy threshold $s_{it}^{j} \geq \zeta$

The availability of a node is expressed as $d_i^j \in D$: $y_t^{i,j} = 1$, if $s_{it}^{j} \geq \zeta$ else $y_t^{i,j}$=0 whenever the node is unavailable during the time slot t. Here, the value of SoC $s_t^{i,j}$ depends on the SoC at the previous time period $s_{i(t-1)}^{j}$, green energy input g_{it}^{j}, and energy consumption e_{it}^{j} at time t. For all $d_i^j \in D$, the SoC $s_{i(t-1)}^{j}$ is expressed as [4], [5], [6], [7]:

$$\forall d_i^j \in D: s_t^{i,j} = min(\delta, max(s_{i(t-1)}^{j} + g_{it}^{j} - e_{it}^{j}, 0)) \quad (1)$$

The surplus green energy for regional node i at time slot t is $(s_{i(t-1)}^{j} + g_{it}^{j} - e_{it}^{j})$. Here, SoC can range between 0 and δ. If the green energy of all nodes is depleted, the brown energy consumption can be calculated as $(e_{it}^{j} - g_{it}^{j} + s_{i(t-1)}^{j})$. The value of e_{it}^{j} is determined based on the functions assigned to d_i^j at t.

The overall energy consumption for all $d_i^j \in$ D considers both direct usage and offloading overhead scenarios [4], [5]. Direct usage is estimated based on the occupied resources of a node for each microservice k, with the energy consumption given by $\frac{\sum_{k=1}^{l}((\alpha_t^{i,j,k}+\beta_t^{i,j,k})\times q_k)}{\Omega}$ multiplied by the power consumption rate υ. Offloading overhead energy consumption is calculated as $\sum_{k=1}^{l}(oe^{k,trans}\times(\kappa_t^{i,j,k} - \alpha_t^{i,j,k})) + \sum_{k=1}^{l}(oe^{k,recv} \times \beta_t^{i,j,k})$. The local and global assignment of functions are denoted by $\alpha_t^{i,j,k}$ and $\beta_t^{i,j,k}$ respectively. The overhead energy cost [4], [5] per microservice for transmitting and receiving are expressed by $oe^{k,trans}$ and $oe^{k,recv}$ respectively. Finally, the total energy consumption is multiplied by the time slot length Δt.

The set of microservices is expressed as $B = \{b^k | k \in [1,l]\}$. The workload generated for a collection of microservices at different time intervals across the regions is represented as $\Lambda = \{\lambda_{it}^{j,k}, | i \in [1,n], j \in [1,m], k \in [1,l], t \in T, \lambda_{it}^{j,k} \in [0, \lambda_{it}^{j,k} max]\}$. The $\lambda_{it}^{j,k} max$ denotes the maximum possible workload that can enter into node i at region j. For execution, each workload comprises a specific amount of processing in Million Instructions (MI) expressed as $\Pi = \{\pi^k, | k \in [1,l], 0 \leq \pi^k \in R\}$. The local controller determines the amount of workload $\sigma_{it}^{j,k} \leq \lambda_{it}^{j,k}$ that can be processed locally in its region. The remaining part of workload $\left(\lambda_{it}^{j,k} - \sigma_{it}^{j,k}\right)$ should be offloaded to the nearest region for processing.

Let $F = \{f_t^{i,j,k}, |i\in[1,n], j\in[1,m], k\in[1,l], t\in T\}$be the set of functions is defined as, here functions $f^{i,j,k}$ perform workload execution for the b^k microservice managed by compute node $d_i$in region j. Each function contains five attributes, i.e., $f^{i,j,k}$= < $f_{type}^{i,j,k}$, $f_{priority}^{i,j,k}$ $f_{at}^{i,j,k}$, $f_d^{i,j,k}$ $f_q^{i,j,k}$> where $f_{type}^{i,j,k}$ denotes the function ID to be invoked, $f_{priority}^{i,j,k}$ represents the priority level (high or low), $f_{at}^{i,j,k}$ denotes arrival time, $f_d^{i,j,k}$ represents the deadline, and $f_q^{i,j,k}$ denotes the resource requirement. The required amount of resources for function deployment is determined as $Q = \{q^k | k\in\{1,l\}, q^k\in[0,\Omega]\}$, where q^k is restricted to the the node's maximum capacity γ.

To support auto-scaling feature in Serverless, $f^{i,j,k}$ may contain multiple function replicas at time t, restricted at R. Multiple replicas will ensure scalability and fault tolerance. Following the replica-level modelling in [4] and [5], we set $\Psi = \{\kappa_t^{i,j,k} | i\in[1,n], j\in[1,m], k\in[1,l], t\in T, \kappa_t^{i,j,k}\in[0,R]\}$ represents the required number of replicas per function. Based on the expected incoming workload, the number of replicas for b^kon d_i^jat t is estimated as $\kappa_t^{i,j,k} = min(R, \lceil \frac{\lambda_{it}^{j,k}\times\pi^k}{q_k} \rceil)$, which may receive a value between 0 to R. In the case where d_i^j is unavailable at t, then $\kappa_t^{i,j,k}$ becomes 0 (i.e., zero replica). To model the replicas, the F is expanded to $F = \{f_t^{i,j,k,x}, |i\in[1,n], j\in[1,m], k\in[1,l], t\in T, x\in[1,\kappa_t^{i,j,k}]\}$, where x represents the x^{th} replica.

Based on the system modeling above, the primary objective of the approach is to minimise the usage of brown energy by maximising the usage of green energy at t time interval. Mathematically, the optimization objective is modeled as:

$$min\sum_t \sum_{i=1}^{n} \sum_{j=1}^{m} (max(e_{it}^j - g_{it}^j + s_{i(t-1)}^j, 0)) \quad (2)$$

Subject to:

$$(\sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_t^{i,j,k} + \beta_t^{i,j,k}) = (\sum_{i=1}^{n} \sum_{j=1}^{m} \kappa_t^{i,j,k}) \forall t\in T, \forall b^k\in B \quad (3)$$

$$\kappa_t^{i,j,k} = y_t^{i,j} \times min(R, \lceil \frac{\lambda_{it}^{j,k}\times\pi^k}{q_k} \rceil) \forall t\in T, \forall d_i^j\in D, \forall b^k\in B \quad (4)$$

$$(\sum_{k=1}^{l} ((\alpha_t^{i,j,k} + \beta_t^{i,j,k})\times q^k)) \leq (y_t^{i,j}\times\omega) \forall t\in T, \forall d_i^j\in D \quad (5)$$

$$f_w^{i,j,k} + f_p^{i,j,k} \leq f_d^{i,j,k} \quad (6)$$

$$SOV(f_t^{i,j,k}, d_i^j) \neq SOV(f_t^{i,j,k}, d^p) \Rightarrow f_t^{i,j,k} \times y_t^{i,j} = 0 \quad (7)$$

For any time period t, Eq. (3) ensures that all function replicas are scheduled, while Eq.(4) ensures that no scheduling of microservices on a node takes place if the node is unavailable. Eq. (5) specifies that the total capacity required for assigned functions must not surpass the maximum available capacity ω of a node, either local or offloaded. Additionally, the incorporation of variable $y_t^{i,j}$ prohibits assignment on unavailable nodes. Eq. (6) ensures that the expected maximum time to finish function execution must not violate the deadline. Here $f_w^{i,j,k}$ is the waiting time for scheduling the function and $f_p^{i,j,k}$ is the processing time for function. Eq. (7) guarantees that functions assigned to any node possess full authority and autonomy over their resources.

However, the Serverless workload scheduling problem is not solvable in polynomial time due to its NP-hard nature. Therefore, we propose a novel stable matching based solution, as it offers an advantage by providing an approximate solution to this highly complex combinatorial optimization problem. The stable matching problem is a generalisation of the stable marriage problem treated in [8].

a.2 Stable matching algorithm

At each region, the function invocation requests are submitted to the local controller. If the region has not sufficient energy for function execution, then the global controller is notified. The allocation of function requests to a node is performed by the local controller, based on the features of the functions, energy and QoS availability of nodes in the region provided through the monitoring component of MAPE-K model. Now we discuss the stable matching algorithm, as shown in Figure 2.15:

Algorithm 1: Matching based Function Scheduling

Input: Set of regions, set of nodes with respective resource capacities, set of functions with their required resource capacities,
Output: A stable matching μ of functions and nodes
while $t \leq T$ **do**
  Phase 1: Initialization:
  /* Regions */
  R $\leftarrow \{R | r_t^{i,j,k} \in R, r_t^i = \psi\}$
  /* Nodes */
  D $\leftarrow \{D | d_i^j \in D, i = null\}$
  /* Resource Capacities */
  Cap $\leftarrow \{Cap | cap_{it}^j \in cap, cap_{it}^j = \gamma\}$
  /* State of Charge */
  S $\leftarrow \{S | s_{it}^j \in S, s_{it}^j = current\ SoC\}$
  /* Functions */
  F $\leftarrow \{F | f_t^{i,j,k} \in F, k = null\}$
  Phase 2: Preference List Construction:
  /* Creating a function's preference list */
  for *each function* $f_t^{i,j,k} \in F$ **do**
    1. Find the candidate node d_i^{pj} such that $(d_i^{pj}(SoC, QoS)) \succ_f (d_i^j(SoC, QoS))$;
    2. Add d_i^{pj} into the preference list of node $f_t^{i,j,k}$.
  end
  /* Creating a node's preference list */
  for *each node* $d_i^j \in D$ **do**
    1. Find the next function $f_t^{i,j,k}$ such that $cap_{it}^{jq} \geq f_{qt}^{i,j,k}, \forall q \in Q$ and $f_t^{i,j,k}$ releases the least amount of resources;
    2. Add $f_t^{i,j,k}$ into the preference list of node d_i^j.
  end
  Phase 3: Matching:
  while $\exists f_t^{i,j,k} \in F$ *is not placed* **do**
    $d_i^j \leftarrow$ Find the top rank in $P(f_t^{i,j,k})$;
    if $cap_{it}^{jq} \geq f_{qt}^{i,j,k}, \forall q \in Q$ **then**
      Assign $f_t^{i,j,k}$ *to* d_i^j;
      $cap_{it}^{jq} = cap_{it}^{jq} - f_{qt}^{i,j,k}, \forall q \in Q$;
    end
    else
      Determine all $f_t'^{i,j,k}$ *to satisfy* $f_t^{i,j,k} \succ_d f_t'^{i,j,k}$;
      Reject all $f_t'^{i,j,k}$ *and mark* $f_t'^{i,j,k}$ *as unengaged;*
      Update node d_i^j *resources as:*
      $cap_{it}^{jq} = cap_{it}^{jq} + f'_{qt}{}^{i,j,k}, \forall q \in Q$
      Eliminate $f_t'^{i,j,k}$ *out of* $P(d_i^j)$;
      Eliminate d_i^j *out of* $P(f_t'^{i,j,k})$;
    end
  end
end

Figure 2.15: Matching based function scheduling in energy-aware Continuum systems

We execute a stable matching algorithm on each local agent. In the *Initialization* phase, the local controller collects the current list of functions with their respective resource demands and QoS, as well as the list of nodes along with their resource availability, latest SoC values. In the *Preference List Construction* phase, we create function's preference list and node's preference list. In the function's preference list, each function's replica $f^{i,j,k,l}$ prefers to be located in the node that has better SoC and satisfies its resource constraints and desired QoS levels. Similarly, in the node's preference list, each node builds a preference list over functions. This preference list is created according to the consolidation policy, in which each node prioritises enhancing resource utilisation by deploying more functions. Additionally, it implies that the node favours assignment over unassignment. The *Matching* phase is inspired by the deferred acceptance algorithm [8]. The matching procedure commences from some unassigned functions $f_t^{i,j,k}$.The function $f_t^{i,j,k}$selects the highest rank node d_i^j from its preference list $P\left(f_t^{i,j,k}\right)$ to propose. If node d_i^j has enough resources to place function $f_t^{i,j,k}$, it will accept the function $f_t^{i,j,k}$. However, if node d_i^j lacks the necessary resources, it rejects $f_t^{i,j,k}$. Before rejecting $f_t^{i,j,k}$, node d_i^j rejects all its matched functions $f_t^{i,j,k}$ such that $f_t^{i,j,k}$is preferred over $f_t^{i,j,k}$ $(i.e., (f_t^{i,j,k}) \succ_d (f_t^{i,j,k}))$. Subsequently, function $f_t^{i,j,k}$ removes node d_i^j from its preference list $P\left(f_t^{i,j,k}\right)$ and restarts the proposing process. Consequently, the algorithm can eventually converge to a stable assignment of functions and nodes within a finite number of steps.

This has served as one of the foundations for our work into energy-aware scheduling; we plan to validate the scalability of our approach and work further going forward.

b) Energy-aware scheduling

The Carbon-aware Model Agent (CMA) is a key component of the AI-Enabled Orchestrator within the COGNIT Serverless platform. Its primary role is to make intelligent scheduling decisions based on the availability of low-carbon energy sources. In modern data centres, the energy consumption required for computational tasks can have a significant environmental impact. Traditionally, scheduling decisions prioritise performance and resource availability. However, carbon-aware scheduling introduces a sustainability dimension by considering the carbon intensity of the energy used during execution.

As mentioned at the beginning of this section, the AI-Enabled Orchestrator leverages several machine learning models to predict workload demands and resource usage. Additionally, it retrieves renewable energy availability forecasts from online sources, such as electricitymaps. This enables the AI-Enabled Orchestrator to dynamically adapt to fluctuating energy conditions. By scheduling functions and Serverless Runtimes to run when and where renewable energy sources, such as wind and solar, are available, the AI-Enabled Orchestrator can potentially reduce carbon emissions while still maintaining optimal application performance. For example, non-latency-sensitive functions can be executed in data centres powered by greener energy sources, aligning execution with periods of peak renewable energy availability. Ideally, high energy-intensive functions

(such as computationally or I/O-intensive functions) should be executed in data centres or edge servers with the lowest carbon intensity. This approach is particularly effective in the Cloud-Edge Continuum, where diverse computing resources—ranging from edge devices and on-premises infrastructure to cloud providers—can be orchestrated seamlessly, ensuring both sustainability and performance goals are met.

b.1 Scheduling methods

Let R be a set of resources (functions or Serverless Runtimes) to be scheduled immediately, where r_i represents a specific resource, and $\hat{e}_i$ denotes the predicted energy usage for r_i. Let $\hat{R}$ be a prediction of resources that will be scheduled in the future, e.g. next 10 minutes. Assume C is a set of clusters available in the continuum, where c_j denotes a specific cluster and g_j represents the momentary CO2 intensity (e.g CO2 intensity) for that particular cluster, and $\hat{g}_j$ denote the predicted CO2 intensity for c_j over the next 24 hours. Both g_j and $\hat{g}_j$ can be obtained directly from the Internet given the location of the cluster, assuming the cluster is not connected to a microgrid with its own power sources (e.g., battery, solar). Another limitation is that the model does not consider the energy efficiency of a particular cluster. We formulate the following scheduling strategies for different scenarios:

b.1.1 Green-First Scheduling: For resources in R, which represent functions or Serverless Runtimes that need to be scheduled immediately, we prioritise scheduling them in clusters c_j with the lowest momentary CO2 intensity g_i without considering the predicted energy footprint e_i of the workloads. This ensures that the current computational tasks are executed in clusters with the least environmental impact in terms of CO2 emissions or energy sourced from non-renewable sources. However, a potential drawback of this approach is that it would saturate the c

luster with the lowest CO2 intensity first, potentially leading to resource contention or performance bottlenecks. Once the lowest-intensity cluster is fully allocated, remaining resources would be scheduled sequentially in clusters with progressively higher CO2 intensity, which may result in suboptimal performance as workloads might not be distributed evenly across available clusters. Additionally, overloading the most environmentally friendly clusters could limit future flexibility in handling incoming workloads. Another drawback is that we may risk scheduling low-energy footprint workloads on low CO2 intensity clusters.

b.1.2. Energy-Intensive Scheduling: High energy-intensive resources in R, such as computationally or I/O intensive tasks, should be scheduled in clusters c_j where the green intensity $g_j$is the lowest. This ensures that resource-heavy tasks are executed in the most environmentally friendly locations, thus minimising their carbon impact. This strategy avoids the issue of scheduling low-energy workloads in low CO2 intensity clusters, but it only accounts for the current set of workloads R, without considering future workloads.

Note that this method is equivalent to the Immediate Scheduling method if R contains only a single workload.

b.1.3 Future-aware Energy-Intensive Scheduling: If future workloads in $\hat{R}$ are known, more optimised scheduling decisions can be made. However, this requires the ability to predict the execution time of specific workloads to be able to take into account the total integrated energy usage. Calculating $\hat{R}$ and estimating execution time depends on application-specific information, such as the execution history of a particular workflow, and cannot typically be determined without domain knowledge. That said, the predictions do not need to be perfect; they only need to be reasonably accurate to optimise scheduling, provided the estimates are not significantly off.

Additionally, the scheduler must plan how cluster capacity is used over time, considering that different workloads have varying execution and completion times. For example, if a cluster has 5 GPUs, the scheduler must account for how long each workload will allocate a GPU and when it will complete, so that it can effectively schedule future workloads. This requires understanding the temporal dynamics of resource usage to avoid idle time or bottlenecks. The scheduler must carefully balance energy intensity, workload duration, and available capacity, ensuring that workloads are allocated efficiently across clusters while considering both current and future resource needs.

b.1.4 CO2-Optimised Scheduling: In this method, the scheduler integrates predictions for both CO2 intensity and workload energy usage to calculate the total amount of CO2 emissions (in kilograms) released into the atmosphere for each scheduling option. By combining these two factors, the scheduler can determine the carbon footprint associated with running specific workloads on particular clusters over time. To do this, the scheduler evaluates the predicted CO2 intensity $\hat{g}_j$(t) for each cluster c_j over the workload's predicted execution time $[t_{0,}\ t_e]$. It then integrates this information with the workload's predicted energy usage rate e_i(t), calculating the total amount of CO2 emitted during the workload's execution using the following formula:

$$\text{Total CO2 emissions} = \int_{t_0}^{t_e} \hat{g}_j(t)\ \hat{e}_i(t)\ dt \qquad \textit{where}$$

$\hat{g}_j(t)$ is the predicted CO2 intensity (e.g., CO2 kg per kWh) for cluster c_j at time t,

$\hat{e}_i(t)$ is the predicted energy usage of workload r_i at time t,

$[t_{0,}\ t_e]$ is the time window over which the workload is executed.

The integration yields the total CO2 emissions for that workload over its entire execution time. The scheduler computes this value for each potential scheduling option across different clusters and time windows. Once the total CO2 emissions are calculated for each cluster and time window, the scheduler can select the option that minimises the overall

carbon footprint. This ensures that energy-intensive workloads are scheduled in the *most* environmentally friendly clusters, during *times* when CO2 intensity is *lowest*.

This method requires:

1. **Accurate CO2 intensity forecasting**: Access to detailed predictions of CO2 intensity $g_j(t)$ for each cluster over time.
2. **Workload Energy usage predictions**: Reliable estimates of the energy consumption $e_i(t)$ for each workload over its execution time.

b.1.5 Deferred Scheduling: This method introduces flexibility by deferring the execution of non-urgent workloads to a later time when the environmental impact is expected to be lower. Instead of immediately scheduling tasks based on current green intensity or workload characteristics, the scheduler strategically delays tasks to align with periods of reduced CO2 intensity. This approach is particularly effective for energy-intensive tasks (e.g., AI training batch jobs) that are not time-sensitive, allowing them to benefit from waiting for more favourable conditions in terms of renewable energy availability.

To prevent indefinite deferral, users must define a deadline by which each workload or function must be completed. The scheduler must balance the benefits of deferring a workload to reduce carbon emissions with the need to meet its deadline. For instance, a highly energy-intensive workload may be scheduled to run in the near future if it aligns with an expected drop in CO2 intensity, but it could be delayed by another deferred, less energy-intensive workload with an urgent deadline that cannot be pushed further. As new workloads enter the system or as conditions change, the scheduler must continuously re-plan deferred tasks, adjusting its strategy in real time to avoid bottlenecks and maximise environmental benefits while ensuring deadlines are respected.

b.1.6 Waste Heat reuse Scheduling: This method leverages the waste heat generated by data centres and computing clusters to maximise energy efficiency. Rather than treating the heat produced during computational tasks as a byproduct to be discarded, the scheduler optimises the placement of workloads in facilities that can capture and reuse this waste heat for purposes such as heating nearby buildings.

In this approach, the scheduler prioritises energy-intensive workloads (e.g., AI model training) for data centres equipped with waste heat recovery systems. By directing these high-computation tasks to such facilities, the energy consumed can be repurposed, significantly reducing the overall environmental impact of the workload.

Energy-intensive tasks are ideal candidates for waste heat reuse, as they produce larger amounts of heat that can be captured and repurposed. The scheduler must be able to identify which data centres or clusters are equipped with waste heat recovery systems, prioritising these facilities for workloads that generate significant heat.

This method can be combined with others, such as CO2-Optimised Workload Planning or Deferred Scheduling, to both reuse heat and reduce CO2 emissions. By capturing and reusing the waste heat from computational tasks, this method improves overall energy

efficiency, reduces the need for additional heating or energy generation, and contributes to a more sustainable computing infrastructure.

b.2 Methodology

To develop and validate the Carbon-Aware Scheduler (CMA), a controlled environment is essential for designing and testing a range of scheduling algorithms. A simulator provides an ideal platform to model and explore different approaches, enabling in-depth evaluation of algorithm performance under various conditions and CO2 intensity scenarios. Generating realistic workloads is crucial to ensure that the scheduler's performance is accurately assessed. This requires creating workloads that reflect typical usage patterns, variability, and resource demands, allowing the CMA to be tested in conditions that closely mirror real-world deployments.

The methodology involves the following key steps:

- **Simulator Development:** Develop a simulator that can model different scheduling environments, enabling controlled testing of various scheduling strategies and their impact on CO2 emissions and energy efficiency.
- **Carbon-Aware Scheduler algorithms:** Develop and explore different scheduling algorithms based on historical CO2 intensity and simulated workload energy usage patterns. This approach enables initial testing and refinement of the CMA's decision-making algorithm.
- **Integration with AI-Enabled Orchestrator**: Integrate the CMA into the AI-Enabled Orchestrator to facilitate real-time, intelligent scheduling across the Cloud-Edge Continuum.
- **Validation in COGNIT Testbe**d: Test and validate the CMA using the COGNIT testbed and the use cases to verify its performance in a live environment. This step involves assessing the scheduler's ability to optimise for reduced CO2 emissions while maintaining workload performance and resource efficiency.

b.3 Datasets

To enable accurate simulation and testing of scheduling algorithms, we utilised diverse datasets that capture both workload energy usage and CO2 intensity. These datasets provide critical insights for developing carbon-aware scheduling methods.

b.3.1 Workload dataset

To generate realistic datasets in the simulator, we utilised the MIT Supercloud Dataset [13]. Below in Figure 2.16 there are three examples of workload traces from this dataset, with the Y-axis representing power consumption in watts (W), illustrating energy usage patterns of different workloads over time. The full dataset comprises 109,747 workload traces, totaling 2.3 TB of data, providing a comprehensive foundation for accurate simulation and evaluation of scheduling algorithms. Note that power utilisation may vary significantly over time, meaning that a scheduler must account for these fluctuations to optimise energy efficiency and minimise CO2 emissions effectively.

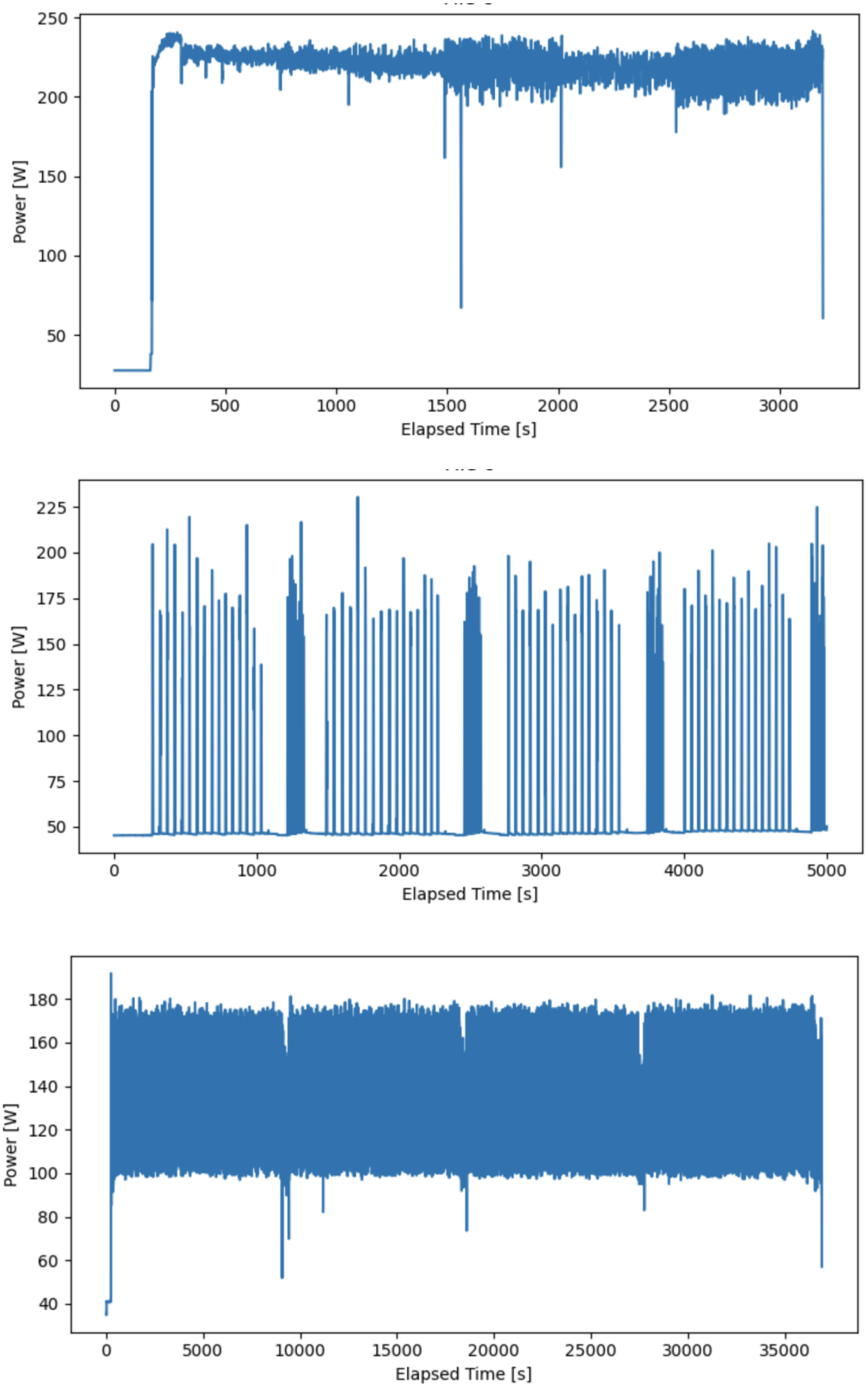

Figure 2.16: Example workload traces from the MIT Supercloud Dataset [13]

b.3.2 CO2 Intensity

The CO2 intensity dataset was generated by downloading data from the ElectricityMap.org website, capturing CO2 intensity values for 73 zones across the European Union over a 5-month period. Figure 2.17 shows examples of CO2 intensity data for Poland and Sweden, highlighting a substantial difference in carbon intensity between different regions.

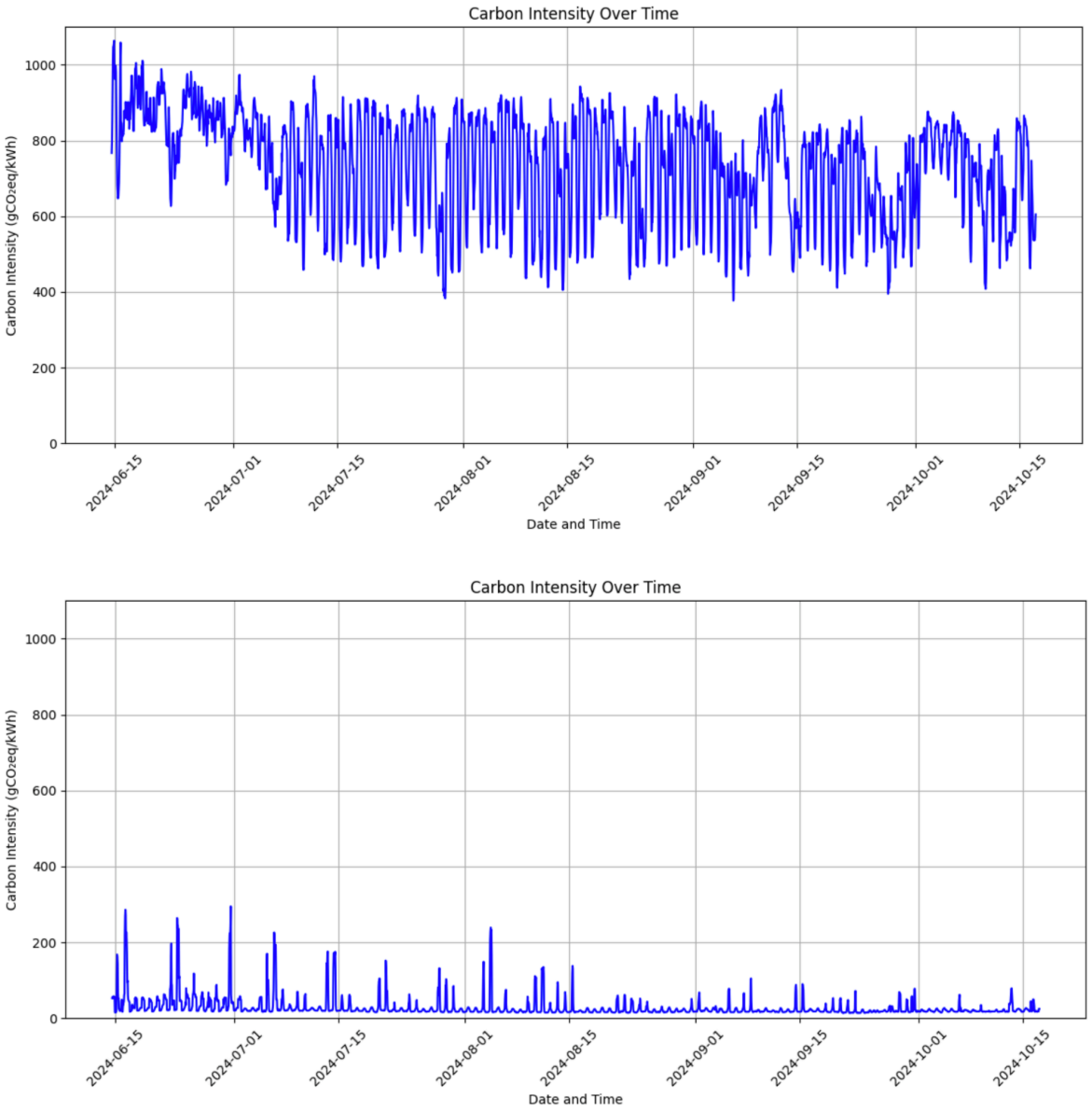

Figure 2.17: Example of how CO2 intensity data can vary between electricity regions: Poland (above) and Sweden (below)

b.4 Simulator

A discrete-event simulator was developed to replay workloads from the MIT Supercloud dataset. To effectively simulate these workloads and energy conditions, the following factors must be considered:

1. **Cluster Generation:** Create edge and cloud clusters across various electricity regions in Europe, each with unique resource configurations (memory, CPU etc.). This setup allows the simulator to model realistic energy variations across clusters.
2. **Pre-Defined Replay Log:** Generate a pre-determined replay log containing a sequence of workflows to be executed. This enables consistent replay of workflow sequences, enabling direct comparisons of different scheduling algorithms.
3. **Workflow Trigger Frequency:** Define the rate at which workflows are triggered from the replay log, allowing control over cluster utilisation levels across different load scenarios. One approach is to randomly sample a wait time from a specified distribution (e.g. normal or beta-distributions), creating variable load patterns. Alternatively, a target utilisation rate can be set, with the wait time calculated to achieve this specific rate.
4. **Modular Scheduler Integration:** Design the simulator to support modular integration of different scheduler algorithms, enabling straightforward experimentation and testing.
5. **Observable Environment for the Scheduler:** Establish an environment that the scheduler can monitor, providing real-time data on cluster utilisation and current CO2 intensities for different regions. This setup enables the scheduler to make informed, adaptive decisions based on up-to-date conditions. Additionally, the simulator should track cumulative CO2 emissions released into the atmosphere, enabling accurate assessment of the environmental impact of different algorithms.

Figure 2.18 shows the clusters currently used in the simulator experiments:

Figure 2.18: Cluster locations currently supported in the Simulator.

Below there is an example of a cumulative emissions time series generated by the simulator. As a baseline scenario, clusters are selected randomly. An effective CO2-aware scheduling algorithm should obviously aim to outperform this baseline and significantly reduce cumulative CO2 emissions. Work is currently ongoing to implement and evaluate the different scheduling strategies.

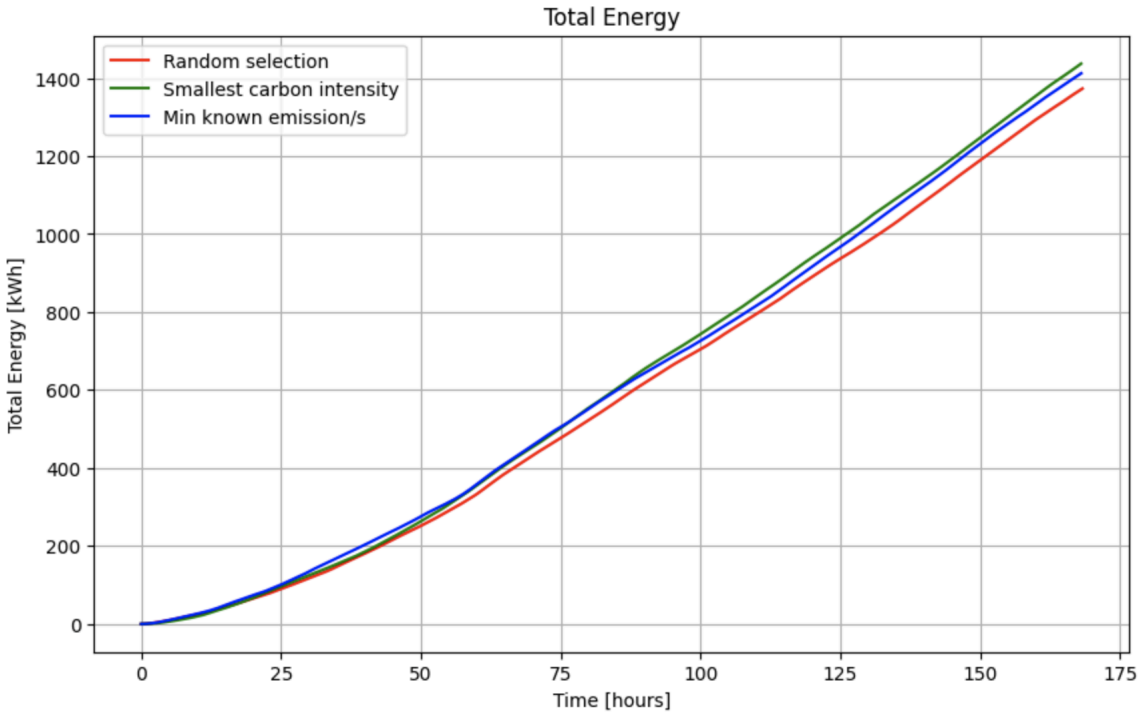

Figure 2.19: Example of cumulative emissions time-series

3. Conclusions and future work

This development cycle focused on advancing workload prediction models and evaluated (Since April 2024) to improve their predictive accuracy and operational efficiency within Cloud-Edge environments. This report highlights the comparison between four core models—LSTM, FFNN, TCN, and GRU—that were reported and integrated into the model repository of the AI-Enabled Orchestrator component of the COGNIT framework. Each model was designed to handle different aspects of workload characterization and prediction, contributing to a more dynamic and efficient orchestration process. LSTM, in particular, was able to capture long-term dependencies in time-series data, leading to its superior performance in predicting metrics like memory usage and disk writes. Meanwhile, the FFNN model, despite being simpler, demonstrated strong performance in CPU usage prediction, indicating its effectiveness in scenarios where sequential dependencies are less critical.

Future development will focus on further refining these models by integrating more advanced architectures, such as attention-based mechanisms and transformers, to handle more complex and dynamic workload patterns. In addition, the next steps will involve expanding the datasets to include additional metrics such as energy consumption and latency, which are becoming increasingly important in optimising resources in the Cloud-Edge continuum. By leveraging real-time adaptive learning, future iterations of these models could dynamically adjust to changes in workload behaviour, enhancing the AI-Enabled Orchestrator's ability to optimise resource utilisation and proactively manage cloud-edge resources. Moreover, the currently developed algorithm for multiobjective resource optimization, i.e., energy-interference aware optimal placement of serverless runtimes, will be implemented, validated, and integrated with the COGNIT Testbed.

To improve the energy-efficiency and sustainability of COGNIT, we formally model a green energy-aware workload scheduling problem for the Cloud-Edge continuum and design a stable matching based approach that considers the availability of green energy nodes and the QoS requirements. These efforts will contribute significantly to achieving more efficient, energy-aware cloud-edge infrastructures, improving overall performance, and enabling more effective, intelligent orchestration strategies within the COGNIT framework.

Finally, in regard to carbon-aware scheduling, our goal is to integrate workload prediction models mentioned in Section 2 (e.g., the LSTM model) to forecast workload utilisation. Using these predictions, we will develop a model to convert utilisation into energy consumption (W), which can then be applied within the simulator described in Subsection 2.2.2. Additional work is also needed to finalise the carbon-aware scheduler, including its integration with the AI-Enabled Orchestrator.

References

[1] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*, 1735-1780.

[2] Patel, Y.S., Townend, P. (2024), "A Stable Matching Approach to Energy Efficient and Sustainable Serverless Scheduling for the Green Cloud Continuum," *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Shanghai, China, 2024, pp. 25-35, doi: 10.1109/SOSE62363.2024.00010.

[3] Autonomic Computing (2006), "An architectural blueprint for autonomic computing", *IBM White Paper*, vol. 31, pp.1-6.

[4] Aslanpour M. S., Toosi A. N., Cheema M. A., Gaire R (2022). , "Energy-Aware Resource Scheduling for Serverless Edge Computing", in *Proc. 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, Italy, pp. 190-199.

[5] Aslanpour M. S., Toosi A. N., Cheema M. A., Chhetri M. B. (2024), "faasHouse: Sustainable Serverless Edge Computing through Energy-aware Resource Scheduling", *IEEE Transactions on Services Computing*, pp. 1-14.

[6] Gu L. et al. (2019), "Energy efficient task allocation and energy scheduling in green energy powered edge computing." *Future Generation Computer Systems*, vol. 95, pp. 89-99.

[7] Xu M. , Toosi A. N., Buyya R. (2021), "A Self-Adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing", *IEEE Transactions on Sustainable Computing*, vol. 6, no. 4, pp. 544-558.

[8] Gale D., Shapley L. S. (1962), "College admissions and the stability of marriage", *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15.

[9] Yadav, M. P., Pal, N., & Yadav, D. K. (2021). Workload prediction over cloud server using time series data. In *11th international conference on cloud computing, data science & engineering* (pp. 267-272). IEEE.

[10] Kumar, K. D., & Umamaheswari, E. (2019). Ewptnn: An efficient workload prediction model in cloud computing using two-stage neural networks. *Procedia Computer Science*, 165, 151-157.

[11] Abouelyazid, M. (2022). Forecasting Resource Usage in Cloud Environments Using Temporal Convolutional Networks. *Applied Research in Artificial Intelligence and Cloud Computing*, 5(1), 179-194.

[12] Mahjoub, S., Chrifi-Alaoui, L., Marhic, B., & Delahoche, L. (2022). Predicting energy consumption using LSTM, multi-layer GRU and drop-GRU neural networks. *Sensors*, 22(11), 4062.

[13] Taye, M. M. (2023). Understanding of machine learning with deep learning: architectures, workflow, applications and future directions. *Computers*, *12*(5), 91.

[14] The MIT Supercloud Dataset, Samsi et al, *arXiv:2108.02037*, 2021

[15] Rastegar, S. H., Shafiei, H., & Khonsari, A. (2023). EneX: An Energy-Aware Execution Scheduler for Serverless Computing. *IEEE Transactions on Industrial Informatics*.

[16] Sampaio, A. M., Barbosa, J. G., & Prodan, R. (2015). PIASA: A power and interference aware resource management strategy for heterogeneous workloads in cloud data centers. *Simulation Modelling Practice and Theory*, 57, 142-160.

[17] Calavaro, C., Russo Russo, G., Salvati, M., Cardellini, V., & Lo Presti, F. (2024, June). Towards Energy-Aware Execution and Offloading of Serverless Functions. In *Proceedings of the 4th Workshop on Flexible Resource and Application Management on the Edge* (pp. 23-30).