TALON

Autonomous and Self-organized Artificial Intelligent Orchestrator for a Greener Industry 4.0

Deliverable

D3.3 Initial Overall E2C AI & Resource Orchestrator

Actual submission date: 02/10/2024

Funded by the European Union

Project Number:	101070181		
Project Acronym:	TALON		
Project Title:	Autonomous and Self-organized Artificial Intelligent Orchestrator for a Greener Industry 4.0		
Start date:	October 1st, 2022	**Duration:**	36 months
D3.1 Architecture & Platform Design Blueprint			
Work Package:	WP3		
Lead partner:	ENG		
Author(s):	Sergio Comella (ENG); Salvatore Cipolla (ENG); Ioannis Pastellas (UBI); Ons Aouedi (UL)		
Reviewers:	Ilias Siniosoglou (MINDS); Georgios Delaportas (PROBO); Alexandros Petropoulos (CERTH)		
Due date:	30/09/2024		
Deliverable Type:	DEM	**Dissemination Level:**	PU - Public
Version number:	1.0		

Revision History

Version	Date	Author	Description
0.1	22/07/2024	ENG	ToC Release
0.2	06/09/2024	ENG, UBI	ENG Contribution, UBI Contribution
0.3	16/09/2024	ENG	Refinement document for review
0.4	18/09/2024	PROBO, CERTH	Document Review
0.5	21/09/2024	MINDS	Quality review check
0.6	27/09/2024	ENG	Addresses review's comments
1.0	02/10/2024	ENG	Final coordinator review before submission

Table of Contents

List of figures

List of Tables

Definitions and acronyms

CA	*Consortium Agreement*
ACLs	*Access Control Lists*
API	*Application Programming Interface*
CA	*Consortium Agreement*
CNI	*Container Network Interface*
DAG	*Directed Acyclic Graph*
DoA	*Description of Action*
DSL	*Domain Specific Language*
DPDK	*Data Plane Development Kit*
eBPF	*extended Berkeley Packet Filter*
EC	*European Commission*
EU	*European Union*
E2C	*Edge-to-Cloud*
GA	*Grant Agreement*
JIT	*Just-In-Time*
KSM	*Kube-state-metrics*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
NFV	*Network Functions Virtualisation*
NLAs	*Node Level Agents*
NN	*Neural Network*
OVN	*Open Virtual Network*
OVS	*Open vSwitch*
PPE	*Personal Protection Equipment*
PC	*Project Coordinator*
RMSE	*Root Mean Square Error*
QoS	*Quality of Service*
NG-SDN	*Next Generation Software Defined Network*
SLOs	*Service Level Objectives*
TC	*Technical Coordinator*
UCs	*Use Cases*
WP	*Work Package*

Disclaimer

Executive Summary

The deliverable *D3.3 Initial Overall E2C AI & Resource Orchestrator* outlines the **initial end-to-end test** of the Edge-to-Cloud (E2C) AI and Resource Orchestrator, a key component of the **TALON project**. TALON, funded under the European Union's Horizon Europe program, aims to develop an **autonomous, self-organized AI orchestrator** that fosters **greener**, **energy-efficient**, and **secure** operations in **Industry 4.0** environments.

The primary goal of the **E2C Orchestrator** is to manage and optimize AI models, datasets, and computing resources in a **dynamic and efficient** manner. It introduces a **zero-touch deployment system** that automates the configuration and operation of AI workloads, significantly **reducing human intervention** and **operational complexity**. A core objective is to **minimize the energy footprint** of AI networks by utilizing **AI-based profiling mechanisms** that adapt to **fluctuating resource availability**. The orchestrator also promotes **reusability** of datasets, algorithms, metrics, and models, ensuring **compatibility** across diverse applications and environments while maintaining **high levels of security and privacy**.

This deliverable presents the **system architecture**, technical decisions, and interactions between key components such as the **Service Orchestrator**, **Network Intelligence**, and **Smart Policy Manager**. These components work together to ensure that AI tasks are deployed optimally across **cloud** and **edge environments**. The deliverable also highlights various **use cases** demonstrating how the orchestrator adapts to **dynamic network conditions**, balances resource demands, and ensures task execution meets **application requirements in real time**.

The orchestration system integrates **Kubernetes** [1] for managing containerized AI services, offering features like **real-time monitoring**, **self-healing**, and **energy-efficiency tracking**. This **technical foundation** ensures **scalability**, **efficient resource usage**, and **robust security**, addressing the critical needs of **Industry 4.0 environments**.

Looking ahead, the deliverable outlines future improvements, including refining **AI models for task deployment** and further enhancing **resource management** and **optimization capabilities**. These developments will support TALON's overarching goals of fostering **greener AI networks**, **reducing energy consumption**, and ensuring **seamless and secure AI operations** across **heterogeneous environments**.

Introduction

1.1 Scope and Objectives

The scope of this deliverable is to provide an outline of the initial version of the E2C AI and Resource Orchestrator within the TALON framework (The repository accessible upon request for credentials is GIT: Talon-orchestrator). The E2C Orchestrator plays a critical role in managing, optimising and orchestrating AI models, datasets and compute resources in a dynamic, energy-efficient manner, while ensuring security and privacy in heterogeneous environments.

The main objectives of this deliverable are to develop an autonomous orchestration layer that enables zero-touch deployment, configuration and operation of the network, AI models and computing resources. This approach aims to reduce operational complexity and minimise human intervention. Another key objective is to reduce the energy consumption of the AI network by optimising resource usage and incorporating AI-based profiling mechanisms that take into account the fluctuating availability of computing resources. In addition, the deliverable aims to promote the reuse of datasets, algorithms, metrics and models to ensure streamlined processes and compatibility across different applications and environments. Security and privacy are also critical, with high-level mechanisms integrated into the orchestrator to ensure robust protection in heterogeneous application environments.

The deliverable will also describe the system's ability to adapt to dynamic network conditions, identify the optimal location for AI processing, and ensure efficient coordination between the network and service orchestrators. This orchestration approach will not only optimise energy efficiency, but also ensure that application requirements are met in real-time.

1.2 Relation to other work packages, tasks and deliverables

This deliverable is closely linked to several work packages (WPs), tasks and deliverables within the TALON project. WP2 defines the system, user and use case requirements that shape the development of the E2C AI and Resource Orchestrator, ensuring that its functionality aligns with the broader system architecture. Task 3.3 builds on the findings of the updated D2.1 and D3.1 to design and implement the components of the Orchestrator. Key inputs to D3.3 include D2.1 which defines the user requirements, D3.1 which outlines the conceptual architecture and component structure of TALON, and D3.2 which details the NG-SDN and network orchestration framework. Together, these deliverables define the use cases, platform architecture and integration points required for the effective implementation of the orchestrator.

1.3 Document Structure

The rest of this document is structured as follows:

Section 1 - Introduction (i.e. this section) outlines the scope and objectives of the deliverable, its relationship to other work packages, tasks and deliverables, and provides an overview of the contents of the document.

Section 2 - System Architecture describes the overall architecture of the E2C AI and Resource Orchestrator. It focuses on the key components, including the Service Orchestrator, the Network Intelligence and the Smart Policy Manager, along with their core features and functionalities.

Section 3 - Technical Specifications explains the technical choices that influence the design and operation of the Orchestrator. It also includes an explanation of the model taxonomy and the metrics used to evaluate performance.

Section 4 - Usage Scenarios and Use Cases presents practical scenarios that demonstrate the use of the Orchestrator. It highlights task deployment and network intelligence scenarios, along with the relevant performance metrics and evaluation criteria.

Section 5 - Conclusions summarises the main findings of the deliverable and provides an outlook on next steps, future improvements and areas for further research and development.

System Architecture

1.4 Overall architecture

The goals of the TALON architecture are to enable zero-touch deployment and operation; to reduce the energy footprint of the whole AI network; to guarantee high-level in security and privacy; to assess and boost the AI Edge-to-cloud performance; to enable reusability of datasets, algorithms, models; to present an AI theoretical framework and finally to boost AI explainability and transparency. These goals will be delivered by an architecture that lies in the pursuit of dynamic scalability, achieved by orchestrating computational and communication resources, AI models, and data, while prioritizing energy efficiency.

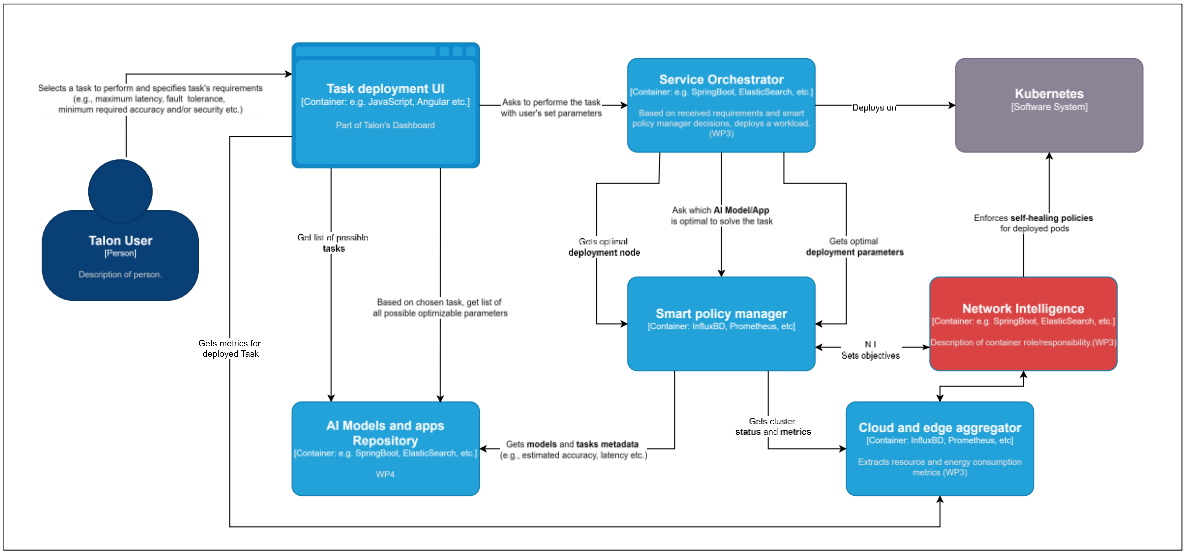


Figure 1. Orchestrator's workflow

The diagram (Figure 1) illustrates the **workflow** of the **TALON orchestration system**, showing how its components interact for **seamless task deployment** and **optimization** across cloud and edge environments.

The process begins with the **Talon User** initiating a task via the **Task Deployment UI** on the **TALON Dashboard**. The user specifies the **task requirements**, which may include factors such as **latency**, **accuracy**, **resource utilisation**, or **energy optimisation**. For example, a user might specify a task that needs to be completed with a high degree of accuracy. The **Task Deployment UI** plays a critical role in guiding the user through this setup by providing a **list of available tasks**. Once the user selects a task, the UI retrieves the necessary parameters for the user to focus on, such as **timing**, **accuracy**, and **energy consumption,** Figure 2 below.

Once the task and the parameters are selected, the **Service Orchestrator** steps in to determine **how and where to deploy the AI Model** to solve the task. Based on the input from the Service Orchestrator and the retrieved models from the **AI Model and Apps Repository**, the **Smart Policy Manager** decides on the **optimal deployment strategy**. It considers several factors: the **current network conditions**, **system resource availability**, and the user-specified requirements for the task. For instance, if **low latency** is critical, the orchestrator might prioritize **edge deployment** to

minimize data transfer time. If **energy efficiency** is more important, it may allocate resources in a way that minimizes energy consumption while still meeting the user's requirements.

Meanwhile, the **Smart Policy Manager** plays a crucial role in **optimising the deployment** by continuously monitoring **real-time cluster data**. It takes into account the **overall system performance**, **resource usage**, and the requirements specified by the user to decide the best allocation of tasks. For example, if network traffic is high in a certain area, the Smart Policy Manager might suggest **moving tasks to less congested nodes**. The Smart Policy Manager is integrated with **Prometheus** and **InfluxDB**, which provide it with **real-time metrics**, including resource consumption and system health indicators. Using this data, the policy manager can adjust deployment strategies **dynamically** to maintain efficiency and ensure the task is executed under **optimal conditions**.

At this time, the **Smart Policy Manager** returns the selected **AI Model** to the **Service Orchestrator** to solve the required task. Then, the Service Orchestrator interfaces with **Kubernetes**, which handles the **actual deployment** of the task to the selected nodes, whether they are in the cloud or at the edge. Kubernetes enforces **self-healing policies** to maintain system resilience, ensuring that any issues with the deployed containers—such as failures or crashes—are **automatically resolved** without user intervention. This **self-healing capability** is a key feature that enables **continuous, uninterrupted service** within the **TALON framework**. In addition to Kubernetes' native policies, **TALON's self-healing** and **self-correcting mechanisms** are described in more detail in **D3.4**.

The **Cloud and Edge Aggregator** works in the background to gather detailed **resource consumption metrics**. This aggregator focuses on the performance of both cloud and edge nodes, providing data on **energy consumption** and **resource utilization**. The information it collects is fed into the **Smart Policy Manager**, enabling further adjustments in task placement or resource allocation. For instance, if edge nodes are consuming too much energy for a particular task, the system might decide to **offload some of the work** to cloud nodes, or vice versa, depending on the **efficiency** and **performance requirements** of the task.

Once the task is deployed, the **Talon User** can monitor its performance through **metrics returned by the system**. These metrics—such as **task completion time**, **accuracy**, or **resource consumption**—are provided through the **Task Deployment UI**, offering full **visibility** into how the task is progressing. This **end-to-end orchestration** ensures that the user's tasks are not only executed optimally but also that the system adapts in **real time** to provide the **best performance** with **minimal resource usage**.

Figure 2. Task Optimization's page

1.4.1 Service Orchestrator

The **TALON Service Orchestrator** is a key component of the TALON architecture, designed to manage the **deployment, orchestration, and execution** of AI services across a wide array of **networked devices**. Its primary role is to simplify and automate the process of deploying AI tasks in environments ranging from **low-power edge devices** to **high-performance cloud systems**. This orchestration mechanism ensures that tasks are deployed in the **optimal location** based on inputs provided by the Smart Policy Manager.

The Service Orchestrator is built to **intelligently balance** resource efficiency, task performance, and energy consumption by considering data from multiple layers of the TALON architecture. It works closely with the **Smart Policy Manager** and **Kubernetes [5]** to ensure that tasks are handled according to the **dynamic needs** of the system. By integrating these elements, the orchestrator can **adapt task placement** based on the available infrastructure, balancing **latency-sensitive workloads** on edge nodes and **high-computational tasks** on cloud nodes. This ensures **low-latency, high-efficiency task execution**, a crucial factor in distributed environments like TALON.

Decision-Making Process

At the core of the Service Orchestrator's functionality is its **decision-making process**. This process begins when a task is submitted via the **Task Deployment UI**, where the user specifies parameters such as **quality**, **speed**, **energy efficiency**, and **cost**. These parameters are passed to the **Smart Policy Manager**, which uses **AI models** specifically trained to handle **multi-criteria optimization**. These AI models evaluate the best possible **trade-offs** based on the user's input and current system conditions.

For instance, if a user prioritizes **energy efficiency** over **processing speed**, the Smart Policy Manager will select a deployment strategy that minimizes energy consumption, potentially by routing

the task to **low-power edge devices**. If the user needs **low-latency** execution, the policy manager might instead prioritize **edge nodes** located close to the data source.

The **AI models** used by the Smart Policy Manager are trained to assess and forecast several key performance indicators (KPIs), including:

- **Task Latency**: The time taken from task submission to execution completion.
- **Energy Consumption**: The total energy used by both **computational resources** and **data transfer**.
- **Resource Utilization**: How efficiently resources such as **CPU**, **memory**, and **network bandwidth** are allocated to the task.
- **Task Success Rate**: A measure of how often tasks complete successfully without needing retries due to failures or resource constraints.

These models are designed using **supervised learning** and **reinforcement learning techniques**, enabling the system to **adapt** based on real-time, historical E2C Cluster's data and historical task performance. Metrics such as **Mean Absolute Error (MAE)** for quality predictions and **Precision/Recall** for task success rate predictions are used to train and evaluate these models. These metrics help the system continuously improve its performance by **learning from past decisions**, making the TALON Service Orchestrator highly **adaptive** and **resilient** in diverse and changing environments.

Realt-Time Optimisation and Adaptation

Once the Smart Policy Manager selects the optimal deployment strategy, the **Service Orchestrator interfaces with Kubernetes** to execute the task. Kubernetes is responsible for container orchestration, handling everything from **scaling resources** to **lifecycle management**. In a **distributed network** like TALON, this means that Kubernetes dynamically adjusts the number of nodes dedicated to the task, ensuring that the task is given sufficient resources to complete efficiently.

Additionally, Kubernetes enforces **self-healing policies** to maintain system resilience. If a container crashes or a node becomes unresponsive, Kubernetes automatically redeploys the task to another node, ensuring **minimal downtime** and **uninterrupted service**. This **self-healing capability** is enhanced by TALON's specific mechanisms for **error detection** and **self-correcting behaviours**, which are detailed in **D3.4**.

The Service Orchestrator's integration with **Prometheus** and **InfluxDB** allows for **continuous monitoring** of real-time metrics, as shown in figure below Figure 3, such as **CPU utilization**, **memory usage**, **task completion time**, and **energy consumption**. These metrics enable the orchestrator to dynamically adapt its decisions as tasks are executed. For example, if the system detects that a particular edge node is becoming **overloaded**, it might shift some of the tasks to **cloud nodes** or **other edge devices**, distributing the load to ensure **smooth system operation**.

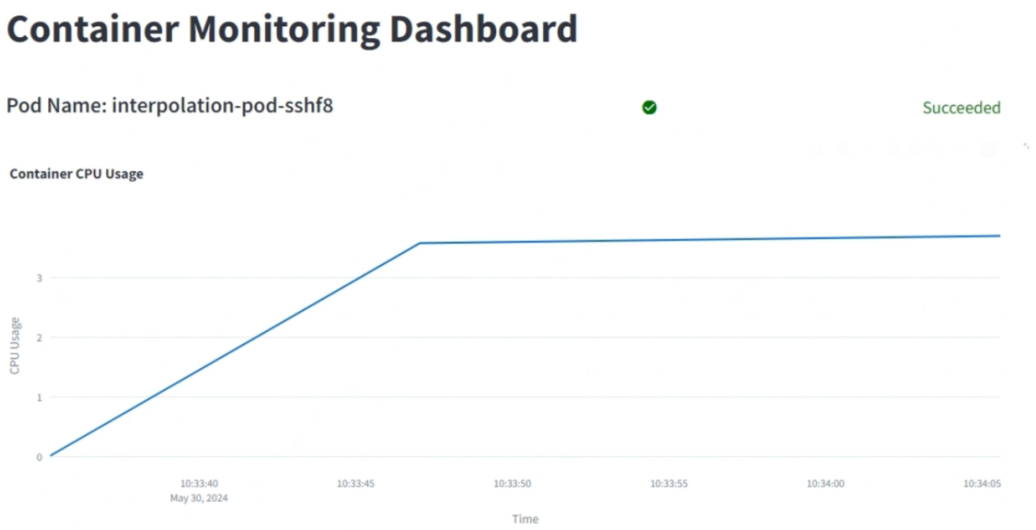


Figure 3. Real-time Monitoring Dashboard

State-of-the-Art Development

The TALON Service Orchestrator is designed to be **state-of-the-art** due to its **AI-driven decision-making** combined with **Kubernetes-based orchestration**. The integration of **advanced AI models**, such as those trained on **deep reinforcement learning**, allows the orchestrator to make **predictive decisions** about resource allocation, latency management, and energy efficiency. This predictive capability is a significant leap forward from traditional static resource allocation methods.

The TALON Service Orchestrator, through its **AI-enhanced decision-making** and **Kubernetes-based automation**, ensures that tasks are optimally deployed across the network. Its real-time monitoring, coupled with **self-healing** capabilities, guarantees both **flexibility** and **efficiency** in task execution, making it a **state-of-the-art solution** for managing AI services in **heterogeneous, distributed environments**.

Key features and capabilities

The **Service Orchestrator** in the TALON framework is a **core component** designed to **optimize the deployment and lifecycle management** of AI services across a **distributed network** of computing resources. It automates the orchestration process by **intelligently allocating tasks** based on **available resources**, **energy efficiency**, and **application requirements**. One of its **key capabilities** is **zero-touch deployment**, which enables the automated management of AI services with **minimal human intervention**. This approach significantly reduces the complexity of managing AI services, allowing the system to operate efficiently and autonomously.

What is Zero-Touch Deployment?

Zero-touch deployment refers to a process where **AI services** and **applications** are automatically deployed, configured, and managed without requiring any **manual intervention** from administrators or users. In this context, the Service Orchestrator uses advanced AI-driven decision-making processes to handle the **entire lifecycle** of task execution, from initial deployment to completion. This includes selecting the **best nodes** for deployment, managing resources dynamically, and applying **self-healing policies** to address any failures or performance degradation in real-time.

To illustrate zero-touch deployment in action, consider the following steps in a **task lifecycle**:

1. **Task Submission**: A user defines the task requirements via the **Task Deployment UI**, specifying factors that wants to optimize, such as energy efficiency rather than quality of results.
2. **AI-Driven Decision Making**: The **Smart Policy Manager** evaluates the task requirements and current system conditions, such as **resource availability** and **network congestion**, to determine the **optimal deployment strategy**.
3. **Automated Deployment**: The **Service Orchestrator** communicates with **Kubernetes** to automatically deploy the task to the most appropriate nodes, without any user intervention. It handles the necessary configurations and optimizations based on **real-time data**.
4. **Dynamic Monitoring and Adjustment**: Throughout the task execution, the orchestrator monitors **resource usage**, **task performance**, and **system health**. If necessary, the orchestrator dynamically re-allocates tasks to different nodes or adjusts resource allocations to maintain optimal performance.
5. **Self-Healing**: In case of any system failure or disruption, **Kubernetes** automatically redeploys the task, ensuring that there is **minimal downtime** and maintaining continuous service availability.

1.4.2 Network Intelligence

This component focuses on the Distributed Intelligence Functions Toolkit, which bridges edge-to-cloud networking mechanisms using Next Generation Software Defined Networks (NG-SDN). The component is responsible of adding networking mechanisms and intelligence on top of the TALON Smart Orchestrator.

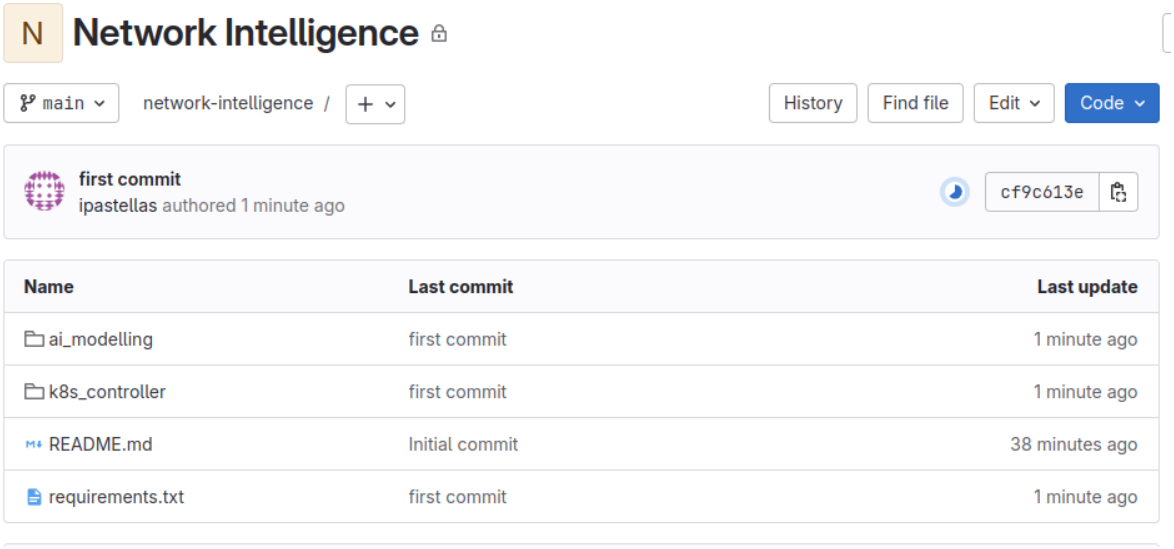
Using, Kubernetes with Kube-OVN as CNI, *b*ridging SDN to Edge and Cloud Native, and enabling different network mechanisms and capabilities, such as VPC, Subnet, customize route, security groups etc. you cannot find corresponding functions in any other CNIs, giving control to processing on the data plane.

The data plane consists of cloud-native application nodes managed by an orchestration engine, while the control plane enforces network policies. The toolkit enables data to be harvested from the data plane and translated into policies that control the network.

Distributed applications consist of containerized pods (the smallest unit of computing in Kubernetes) that communicate with each other or external entities. These pods can be managed in real-time through policies that specify triggering conditions and actions, which can affect traffic management (e.g., traffic-dropping, allowance, or tagging) at various OSI layers (Layer 3, 4, and 7), as discussed within D3.2.

In addition, this component serves as the controller that applies different network policies.

Below, there is a Gitlab screenshot with all code for the Network Policy Controller, and the AI modelling on pod behavioural data, for enhancing existing network policies.



1.4.2.1 Key features and capabilities

This component provides several key functions of the network orchestration framework, including:

- In-line and off-line data processing from the data plane
- Virtual Logical Network topology
- Traffic replication for further analysis
- Enforcement of network policies based on decisions in the control plane
- Use of AI, in the design and the enforcement of the network policies, utilizing logged monitoring data.

The TALON framework integrates both edge and cloud pods, breaking down policies into specific conditions and actions across the OSI layers. These are converted into binary artifacts based on the underlying hardware, supporting both **eBPF** [11] acceleration and other programmable network interfaces. The TALON orchestrator centrally manages the translation and deployment of these actions through control-plane signalling. Furthermore, using AI and Machine Learning on logged monitoring data about the behaviour of nodes and pods, TALON incorporates AI-assisted network policies to enhance the Intelligence of this component.

The policies defined:

- A vertical scaling policy in Kubernetes automatically increases the resources (e.g., CPU, memory) allocated to a pod when high CPU usage is detected, ensuring optimal performance under heavy load.
- A Quality of Service (QoS) policy allocates additional network bandwidth to pods in response to increasing demand, prioritizing resource allocation for high-traffic or critical workloads.
- Enhanced by AI, these policies leverage historical and behavioural data to predict future CPU spikes or network demand, enabling proactive enforcement of vertical scaling and QoS adjustments before bottlenecks occur

These policies optimize TALON operations by ensuring efficient resource use, real-time scalability, and minimal disruptions in dynamic environments.

1.4.3 Smart Policy Manager

The containers require resource management to ensure they operate efficiently, without overusing energy or overwhelming system resources. The Smart Policy Manager task aims to tackle the challenges of energy optimization in containerized environments by using a machine learning approach to dynamically predict and enforce optimal policies. By leveraging time-series data on resource usage, including CPU and memory consumption as well as energy (joules) metrics, this project applies machine learning models to predict policies such as scaling up/down resources, task offloading, or data offloading. This allows for automated and dynamic management of resources, improving efficiency and reducing energy consumption.

The dataset for this task comes from energy and resource usage running on cloud infrastructure monitoring through Kepler [3] tool. It contains various metrics that track resource consumption over time, allowing us to observe trends and patterns in container behaviour. For model training, we used several input features such as "*cpu_usage"* (CPU consumption by the container), "*memory_usage"* (Memory consumption by the container), "*system_cpu_usage"* (CPU usage of the entire system), "*user_cpu_usage"* (CPU usage by the user), "*core_joules"* (Energy consumption in the core of the system in joules), "*dram_joules"* (Energy consumption in DRAM), "*total_joules"* (Total energy consumption), *package_joules* (Energy consumption of the package).

The preprocessing steps:

The container ID was one-hot encoded to allow for unique identification of each container during model training. The numeric features were standardized using StandardScaler to ensure they follow a normal distribution, making the training process more efficient.

Policies were applied based on thresholds of resource consumption. These policies include:

- *"Scale_up"*: When the container exceeds resource thresholds (e.g., CPU usage > 90th percentile).
- *"Scale_down"*: When the container operates below a resource threshold (e.g., CPU usage < 20th percentile).
- *"Task_offloading"*: When the container approaches resource overload but hasn't yet crossed the threshold.
- *"Data_offloading"*: When energy consumption exceeds a defined limit.

The project employs a Long Short-Term Memory (LSTM) model, a type of recurrent neural network (RNN), to predict policies based on time-series data. LSTMs are particularly suited to this problem because they can capture dependencies and patterns in sequential data, such as resource consumption over time.

Several studies validate the use of LSTM models for such problems. For example, LSTMs have been extensively applied in **time-series forecasting** for traffic and mobility prediction [9]. Similarly, research in fields like **financial market predictions** and **energy consumption forecasting** has demonstrated the model's effectiveness due to its ability to handle large, sequential datasets [10].

Model Architecture

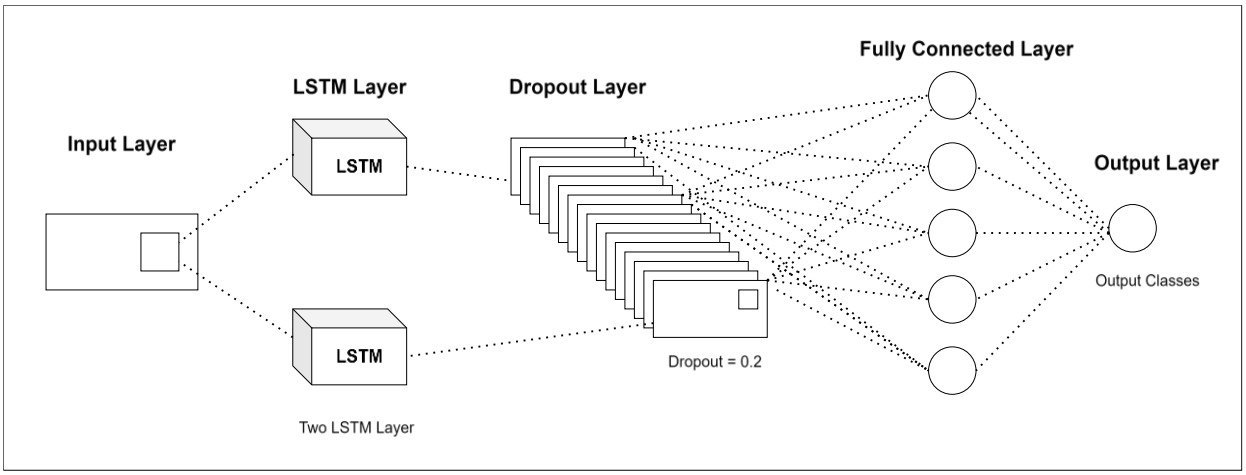


Figure 4. LSTM Model architecture

The LSTM model, represented in Figure 4, is composed of the following:

- Input Layer: Accepts sequences of 10-time steps (window size) for each feature.
- LSTM Layers: Two LSTM layers with 32 hidden units each, which capture temporal dependencies.
- Dropout Layer: A dropout layer with a rate of 0.2 to prevent overfitting.
- Fully Connected Layer: A dense layer that outputs the predicted policy (classification into one of the defined policies).

The model parameters are as follows:

- Input size: Number of features (including CPU, memory, energy metrics).
- Hidden size: 32 units in each LSTM layer.
- Number of layers: 2.
- Output size: Number of possible policies (5 in total: scale_up, scale_down, task_offloading, data_offloading, no_action).

1.4.4 Key features and capabilities

The Smart Policy Manager can automatically adjust system policies in real-time based on changing container resource usage patterns. By analysing time-series data, it predicts optimal policies such as scaling resources up/down, task offloading, and data offloading to manage system load effectively. In particular, leveraging the LSTM model, the system predicts resource allocation adjustments that help optimize energy consumption. The ability to minimize power usage in response to fluctuating CPU, memory, and energy metrics ensures that cloud infrastructure operates efficiently without overloading or underutilizing resources. The system uses the Kepler tool, which provides accurate, real-time energy consumption metrics from containers. This allows for precise tracking of CPU, memory, and energy usage, which is critical for making intelligent policy decisions. Moreover, users can define their own thresholds for resource utilization, energy consumption, and system overload conditions. This customization enables tailoring the Smart Policy Manager to specific organizational needs and different containerized environments. In addition, the integration of the *CodeCarbon[8]* library tracks the CO2 emissions generated during model training. This helps developers monitor and optimize the environmental impact of AI operations, supporting eco-friendly practices in cloud infrastructure management.

Below (Figure 4) there is the accuracy and loss results of the LSTM model during the training. These two figures together show that the LSTM model improves steadily with more training epochs, reaching

a good balance between training performance and generalization to unseen data, crucial for the Smart Policy Manager task.

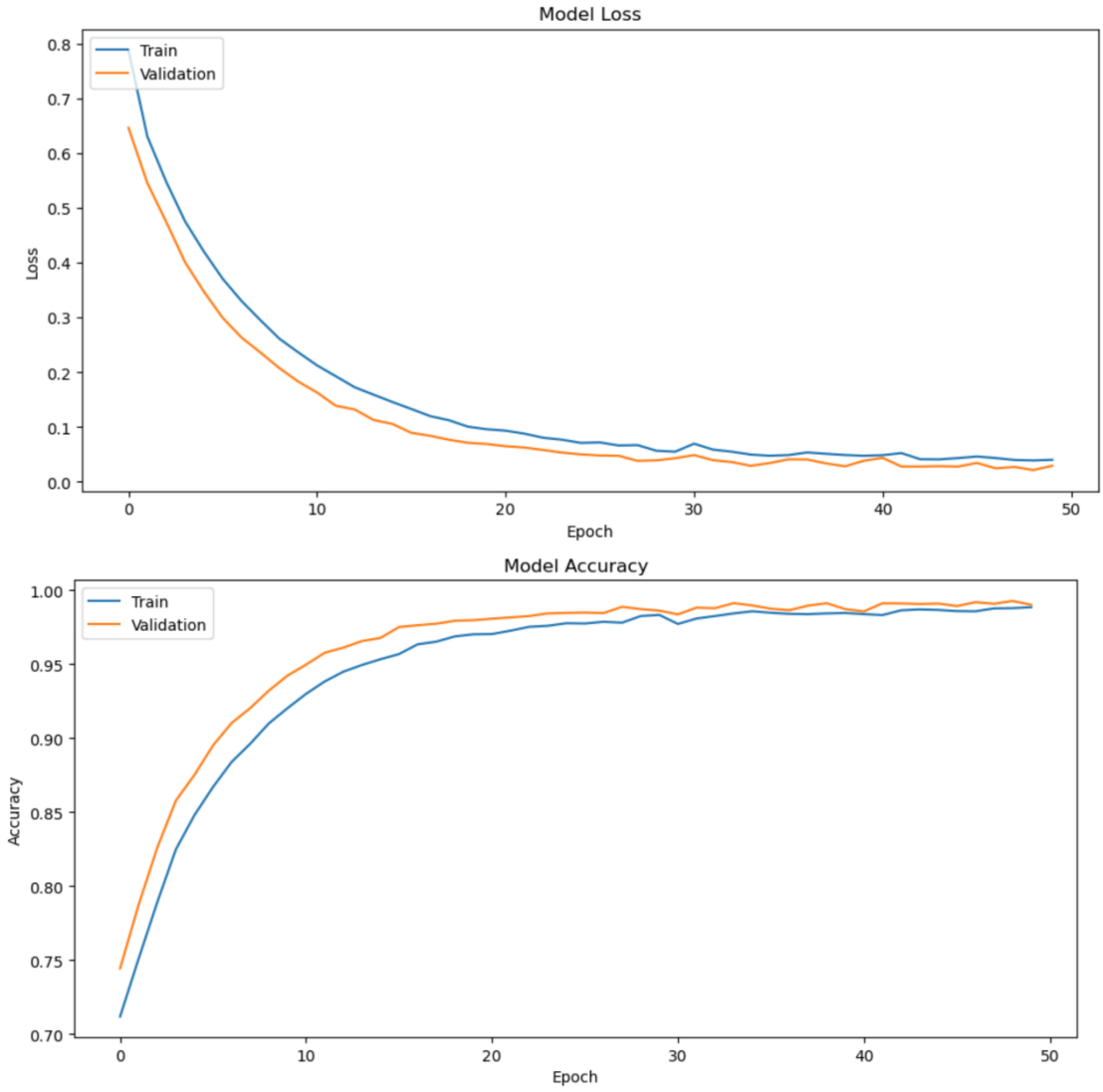


Figure 4. Train/validation accuracy and loss

This table presents the performance metrics of the LSTM model. Overall, the results demonstrate that the LSTM model is both accurate and efficient, making it fit well for the Smart Policy Manager task.

Task	Name	Dataset	Mae	Acc	Prec	F1	Exec time	Energy cons	CPU usage
Policy prediction	LSTMmodel	Kepler data	NA	98.92	99.05	98.97	0.002757	0.0018	0.0015

Table 1. LSTM results

To compare the LSTM model to other implementations:

- LSTM Model: Achieves 98.92% accuracy, 99.05% precision, and 98.97% F1 score on the Kepler dataset, with low execution time (0.0027 seconds) and energy consumption.

- Traditional Models (e.g., Decision Trees, SVM): Typically achieve lower accuracy (~90–95%) and are less efficient in handling sequential data like time-series, which reduces their precision and F1 score in such tasks. They often consume more energy and resources.
- Other Deep Learning Models: While they may reach similar accuracy, they struggle with time-series data, where LSTM excels due to its ability to handle long-term dependencies.

Thus, the LSTM model is optimal due to its high accuracy, precision, and efficiency in managing time-series data, making it superior for real-time policy prediction tasks.

1.4.5 Algorithm selection

The process of selecting the optimal algorithm for a specific task in the Talon project's smart orchestrator involves a systematic evaluation of available algorithms based on user-defined priorities. The orchestrator ensures that various performance characteristics, such as quality, speed, energy efficiency, and cost, are taken into account when making this selection.

Process Overview

This selection process is designed to compare multiple algorithms based on a set of pre-calculated metrics. Users can specify which parameters they wish to optimize, guiding the orchestrator to make an informed choice based on their priorities.

Detailed Steps

1. Expected Inputs:
    - List of Available Algorithms: A list of algorithms capable of solving the given task.
    - List of Chosen Parameters: User-specified parameters that define what needs to be optimized (e.g., quality = 70/100, speed = 30/100). These parameters indicate the user's priorities in selecting an optimal algorithm.
2. Normalization of Metrics:
    - Each algorithm has several associated metrics that have been calculated using a common dataset to ensure comparability.
    - For every algorithm, the metrics are normalized. Normalization is crucial because different metrics can have varying units or scales. This step scales all values to a common range, making them suitable for direct comparison.
3. Conversion to Relevant User Input:
    - For each algorithm, convert the normalized metrics into a form that reflects their relevance to the user-specified parameters. For example, metrics related to speed, energy consumption, or quality are mapped to the user's input values.
    - This conversion helps translate raw metrics into a score or value that indicates how well the algorithm meets the user's criteria.
4. Calculation of Coefficients:
    - Using the converted values, calculate a coefficient for each algorithm. This coefficient represents how closely an algorithm aligns with the user's desired optimization parameters.

- The coefficient is a weighted aggregation of the relevant metrics, with weights assigned based on the user-defined importance of each parameter (e.g., if quality is most important, its corresponding metric will have a higher weight).

1. Sorting and Selection:
   - The coefficients of all algorithms are then sorted in descending order. The algorithm with the highest coefficient is considered the most likely to fulfil the user's requirements and is selected as the optimal choice.
   - This selection at the moment utilizes ORTools (Google, 2024) from Google. While this might be overengineered currently, future versions might exploit the power of Linear integer programming to better integrate this algorithm selection with resources specifications (e.g., how much CPU/RAM reserve to a specific service).

Example of the Process in Action

- A task has two potential algorithms (linear interpolation, pad interpolation) with various metrics like accuracy, time of execution etc.
- The user specifies that they want to prioritize quality (70%) and energy efficiency (30%).
- For each algorithm, the relevant metrics are normalized, linked to the user's set parameters, and merged to calculate a coefficient.
- The coefficients are then sorted, and the algorithm with the highest coefficient is selected by the orchestrator as the optimal choice for deployment.

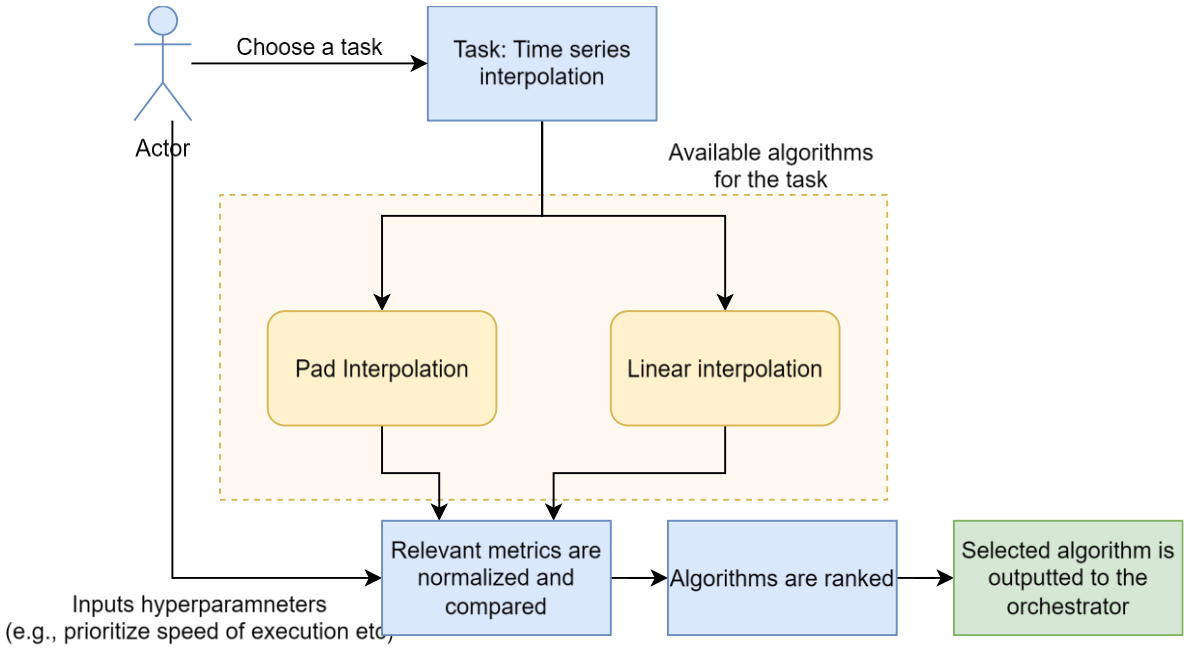


Figure 5. Process in action

By considering user-defined priorities and normalizing algorithm metrics, this process ensures that the chosen algorithm is best suited to the user's specific needs.

Technical Specifications

This chapter covers the core technical decisions and methods behind the TALON E2C AI and Resource Orchestrator. Kubernetes was selected as the orchestration platform for its ability to seamlessly manage edge and cloud environments while ensuring scalability and resource efficiency. Prometheus, Kepler, and InfluxDB provide real-time monitoring and energy tracking, aligning with TALON's objectives of zero-touch deployment and energy optimisation.

The orchestration system dynamically selects the most appropriate method for task execution, optimizing factors like accuracy, speed, and energy consumption based on user-defined criteria. This ensures tasks are handled efficiently while balancing performance and resource use.

1.5 Technical Decisions

The decision to adopt **Kubernetes** (The Linux Foundation, 2024) as the core orchestration platform is motivated by its ability to seamlessly connect *edge and cloud environments*, promoting efficient resource management and scalability. A major advantage of this approach is Kubernetes' **robust monitoring stack**, which offers real-time visibility into the ecosystem. This includes **Prometheus** [2] for performance metric collection, **Kepler**[3] for monitoring energy efficiency, and **InfluxDB**[4] for storing long-term data.

Prometheus [2] tracks AI models and infrastructure health, enabling proactive management and fine-tuning. **Kepler** [3] plays a critical role in monitoring energy consumption, optimizing resource usage, and minimizing the energy footprint of AI tasks. **InfluxDB** [4] stores the collected data and supports long-term trend analysis to ensure AI models are continuously optimized and effectively deployed. Together, these tools provide a **comprehensive view of orchestrator operations**, enabling dynamic training, enforcement, and resource management across both *edge and cloud environments*.

This technical choice aligns with the **TALON project's core objectives** of zero-touch deployment, energy efficiency, and scalability. For example, **Objective O1**, which focuses on enabling *zero-touch deployment and operations*, is well supported by **Kubernetes' automation capabilities** for managing, scaling, and deploying containerized AI services in diverse environments. Its **self-healing mechanisms** ensure resilience, effectively addressing the challenges posed by fluctuating resource conditions outlined in the project's objectives.

Regarding **Objective O2**, which aims to reduce the *AI network's energy footprint*, **Kepler's real-time energy monitoring** is critical. Kepler's tracking of energy consumption across *edge and cloud nodes* enables dynamic adjustments, directly contributing to the project's energy efficiency goals.

In addition, **Objective O4**, which focuses on *improving AI performance assessment and optimization*, is supported by the integration of **Prometheus** and **InfluxDB** for comprehensive, system-wide monitoring and data collection. These tools provide **detailed insights** into performance metrics, enabling data-driven decision-making and continuous AI optimization to achieve the goals of *reducing latency and improving task execution*.

In conclusion, the adoption of Kubernetes as the core orchestration platform plays a pivotal role in achieving the TALON project's objectives of **zero-touch deployment**, **energy efficiency**, and **scalability**. Its seamless integration of edge and cloud environments, combined with a robust monitoring stack featuring Prometheus**,** Kepler**,** and InfluxDB, ensures real-time visibility, proactive management, and long-term optimization of AI models. This strategic choice supports not only the automation and resilience required for diverse environments but also contributes to critical goals like reducing energy consumption and improving AI performance, ultimately driving the success of the project.

1.6 Model Taxonomy Metrics Explanation

The document outlines several methods for handling various tasks, focusing on optimizing specific performance metrics such as **quality**, **speed**, **energy consumption**, and **execution time**. Each method is designed to prioritize different aspects of task execution, allowing users to choose the most appropriate method based on their **optimization requirements**. For example, certain methods may emphasize **accuracy** or **precision**, while others prioritize **faster execution** or **lower energy consumption**.

The table below provides an example of two methods for **time series interpolation**: **Linear interpolation** and **Padding**. Key metrics such as **Mean Absolute Error (MAE)**, **Accuracy (Acc)**, **Precision (Prec)**, **F1 Score (F1)**, **Execution Time**, **Energy Consumption**, **CPU Usage**, and **Maximum Memory Allocation** are included to help assess the performance of each method. Not all metrics are available for every algorithm (they are indicated in the table below as N/A); this might be due to the fact that some measurements were not made or that that specific metric is not relevant for the task. Other metrics (e.g., CPU usage) are directly inferred from Kepler statistics.

Task	Name	Dataset	Mae	Acc	Prec	F1	Exec time	Energy cons	CPU usage	Max mem alloc
Time series interpolation	Linear	Delhi ts	0.021	NA	NA	NA	0.001240	NA	NA	NA
	Pad	Delhi ts	0.028	NA	NA	NA	0.000410	NA	NA	NA

Table 2. Model Taxonomy's table example

Explanation of Metrics

- **Mean Absolute Error (MAE)**: MAE is used to measure the average magnitude of errors in a set of predictions, without considering their direction. It is calculated as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of predictions.

- **CPU Usage and Maximum Memory Allocation**: These operational metrics track the computational resources for real-time processes. They are not available since the measurement are performed after deployment for this specific example.

- **Execution Time**: The time it takes for the method to complete the task.

- **Energy Consumption**: This is the amount of energy used to execute the task. Since this example does not track energy usage, it is marked as N/A.

- **Accuracy (Acc)**: **Not applicable** for the tasks in this example, as **accuracy** is a metric typically used for **classification tasks [6]**, where predictions are compared to categorical labels to determine how many instances were correctly classified. In the case of interpolation, which is a **regression task**, accuracy is not meaningful because the goal is to predict continuous values rather than classify instances into discrete categories. However, for classification tasks, accuracy can be calculated as:

$$Accuracy = \frac{Numer\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

- **Precision (Prec)** and **F1 Score (F1)**: Similarly, **not applicable** for the regression tasks like interpolation considered in this example. Precision and F1 score are typically used in **classification tasks**, particularly where class imbalance is an issue. Precision measures the **accuracy of positive predictions**, while the **F1 Score** balances precision and recall. Though these metrics are irrelevant for interpolation, their formulas are included for reference:

$$Precision = \frac{TP}{TP + FP}$$

  where: TP is True Positives and FP is False Positives.

- The **F1 Score** is the harmonic mean of precision and recall, providing a balanced measure of both metrics:

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

  Where Recall is:

$$Recall = \frac{TP}{TP + FN}$$

For interpolation and other regression tasks, metrics like **Mean Absolute Error (MAE)** or **Root Mean Square Error (RMSE)** are typically more appropriate, as they measure the differences between predicted and actual values on a continuous scale, unlike the discrete predictions of classification tasks.

Usage Scenarios and Use Cases

1.7 Example Scenarios of Orchestrator

This section explores practical use cases where the orchestrator is deployed to manage tasks efficiently across the network. The scenarios demonstrate how various orchestration mechanisms and components interact to ensure the optimal execution of tasks and network operations.

1.7.1 Task Deployment Scenario

Figure 1, illustrates the interaction between the components of the orchestrator. Below, a typical task deployment scenario within TALON's AI orchestration system is described:

1. **Task Selection**: The user begins by selecting a task from a predefined list, which may include options such as **Data Casting**, **Outlier Removal**, or **Interpolation**. This is done through the **Task Deployment UI**, where the user also defines specific parameters for the task, such as **quality** (e.g., prioritizing accuracy), **speed**, or other optimization criteria.
2. **Configuration Interface**: After selecting a task, the system presents a **configuration interface** that allows the user to adjust the task's optimization parameters. For example, if the user selects **quality** as the priority, the system will aim to minimize metrics like **Mean Absolute Error (MAE)** during execution, while possibly deprioritizing speed.
3. **Service Orchestrator and Lifecycle Manager Collaboration**: Once the parameters are set, the **Service Orchestrator** collaborates with the **Lifecycle Manager**, which works in tandem with the **Smart Policy Manager** to determine the most appropriate algorithm for the task. The decision is based on real-time data from the system's infrastructure, such as network conditions, resource availability, and the user's priorities.
4. **Pod Deployment**: Based on the input from the **Smart Policy Manager**, the system deploys a **pod** to execute the task. This pod operates in a cloud or edge environment, depending on the task's requirements and resource availability.
5. **Real-Time Metrics**: Throughout the task execution, the system displays **real-time metrics**, such as **Mean Absolute Error (MAE)**, **execution time**, and other relevant performance indicators. These metrics provide visibility into how well the task is performing and allow the system to adjust the execution dynamically if needed.

Performance Metrics and Evaluation

Once a task is deployed, the system continuously monitors and evaluates its performance using a variety of metrics. For quality-focused tasks, the **Mean Absolute Error (MAE)** metric is critical in assessing the accuracy of the algorithm being used. In addition to **MAE**, execution time is also tracked to provide a balanced assessment of both accuracy and efficiency.

The system uses these metrics to evaluate how well the selected algorithm meets the **pre-defined optimization criteria**. For example, if a **linear algorithm** is chosen for a quality-focused task, the system will record and display the corresponding performance data, including MAE and execution speed, for review. This allows for continuous feedback on the system's decision-making and ensures that the task is executed efficiently based on the user's specified priorities.

1.7.2 Network Intelligence Scenario

In the Network Intelligence, a typical scenario is the QoS management. These policies are created using Kubernetes Kube-OVN supported annotations, which allow administrators to detail the ingress and egress bandwidth limits for targeted pods. For instance, an annotation for setting a maximum bandwidth might look like this:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gateqay-api
  namespace: talon-policies
  annotations:
    ovn.kubernetes.io/ingress_rate: "3000"
    ovn.kubernetes.io/egress_rate: "100"
spec:
  containers:
  - name: qos
    image: docker.io/library/nginx:alpine
```

Figure 6. Pod example of bandwidth limits

In modern Kubernetes (k8s) environments and general modern infrastructures, the enforcement of Quality of Service (QoS) policies is crucial for efficient network resource management. Traditional methods often rely on reactive measures, which can lead to suboptimal performance and resource allocation. Leveraging AI models trained on historical data, this section outlines a proactive approach to predict future bandwidth demands and enforce network policies dynamically using Kube-OVN.

Thus, building on the previous section, this section describes the enhanced above QoS policy. By enhanced, we mean to proactively enforce the QoS policy ahead of its needed enforcement with the assistance of an AI model that is fitted on historical and periodical data of a pod.

The AI-fueled approach utilizes machine learning models trained on historical network usage data of pods to forecast future bandwidth requirements. This predictive capability enables the Kubernetes cluster to apply QoS policies in advance, thereby maintaining optimal network performance and resource utilization.

1.7.2.1 Performance Metrics and Evaluation

The performance of the network intelligence component is done by measuring the overall Kubernetes network resource management. This is done by measuring different metrics such as **CPU utilization**, **overall network latency**, **number of failed pods** and others.

In addition to the that, different ML metrics are used for measuring the performance of the AI/ML models. For the ML models we used **Accuracy**, **Precision**, **Recall**, **F1-score** for classification models (CPU high load binary classification, Network high bandwidth binary classification, etc.) and Mean Squared Error (MSE) for Regression models.

1.7.3 Future improvements

Future enhancements could include refining the AI models used by the **Smart Policy Manager** to better account for a wider range of trade-offs between quality, speed, energy efficiency and cost. This would allow the system to provide even more precise algorithm recommendations based on specific user requirements. In addition, the user interface could be enhanced to provide more detailed real-

time feedback, allowing users to immediately see the impact of their parameter adjustments on overall task performance and resource consumption. Extending these capabilities would further optimise task deployment and improve system usability.

Conclusions and Future Outlook

The Initial Overall E2C AI & Resource Orchestrator is a crucial step towards realising TALON's vision of a dynamic, energy-efficient and secure orchestration framework for Industry 4.0. Throughout the development and integration phases, the orchestrator has demonstrated its ability to efficiently manage AI models, data sets, and compute resources across cloud and edge environments. By leveraging zero-touch deployment, AI-based profiling, and self-organising capabilities, the orchestrator has successfully minimised operational complexity and reduced the energy footprint of AI networks, in line with TALON's core goals.

The **integration of Kubernetes**, coupled with advanced monitoring tools such as Prometheus and Kepler, has proven effective in maintaining scalability and efficient resource utilisation. The orchestrator's adaptability in real-time task deployment and resource allocation ensures that performance is optimised without sacrificing energy efficiency or security. The system's ability to automate resource management based on dynamic network conditions is a key achievement that will drive further advances in AI network orchestration.

Looking forward, several **future improvements** have been identified to enhance the system's capabilities. Refining the AI models used for task deployment and policy management will allow for even more precise decision-making. This includes better handling of trade-offs between quality, speed, energy efficiency, and cost, providing more tailored algorithm recommendations. Furthermore, improving the user interface to offer more detailed real-time feedback will enable users to better understand the impact of their decisions, improving overall usability.

Moreover, expanding the system's integration with additional **next-generation technologies**, such as **AI-enhanced network intelligence** and more sophisticated predictive capabilities, will further strengthen the TALON orchestrator's role in enabling greener, more sustainable AI operations. These enhancements will contribute to TALON's goal of supporting **autonomous and self-organized AI networks** that can seamlessly operate across heterogeneous environments, meeting the demands of Industry 4.0.

With ongoing development and continuous refinement, the orchestrator will remain at the forefront of driving innovation in AI orchestration, supporting the long-term goals of the TALON project and paving the way for a sustainable future in Industry 4.0.

References

[1] **The Linux Foundation**. (2024). *Kubernetes: The Complete Guide to Orchestration*. Retrieved from https://www.linuxfoundation.org/kubernetes

[2] **The Linux Foundation**. (2024). *Prometheus: Monitoring Systems and Services*. Retrieved from https://www.linuxfoundation.org/prometheus

[3] **Cloud Native Computing Foundation**. (2024). *Kepler: Energy Efficiency Monitoring for Cloud and Edge*. Retrieved from https://www.cncf.io/kepler

[4] **InfluxData**. (2024). *InfluxDB: Time Series Data Platform for Metrics and Events*. Retrieved from https://www.influxdata.com/influxdb/

[5] **Burns, B., & Kelsey, H., & Beda, J** (2022). *Kubernetes: Up & Running - Dive into the Future of Infrastructure*. O'Reilly Media.

[6] **Ruben, van den G., Marteen, van S., & al (2022).** The harm of class imbalance corrections for risk prediction models: illustration and simulation using logistic regression. Retrieved from https://academic.oup.com/jamia/article/29/9/1525/6605096

[7] **Wang, C., et al.** (2020). *Energy-Efficient Orchestration for Edge-Cloud Resource Allocation*. *Future Internet*, 12(9), 155. Retrieved from https://doi.org/10.3390/fi12090155

[8] **Clever Cloud**. (2021). Code Carbon. Retrieved from *CodeCarbon.io*

[9] **Yuxiu Hua, Zhifeng Zhao, Rongpeng Li, Xianfu Chen, Zhiming Liu, and Honggang Zhang.** (2024). Deep Learning with Long Short-Term Memory for Time Series Prediction. Retrieved from https://ar5iv.labs.arxiv.org/html/1810.10161

[10] **Greg Van Houdt, Carlos Mosquera & Gonzalo Nápoles.** (2020). A review on the long short-term memory model. Retrieved from https://link.springer.com/article/10.1007/s10462-020-09838-1

[11] **eBPF.io.** (2024). eBPF. Retrieved from https://ebpf.io/

Funded by
the European Union

This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101070181