



> DESIRE6G <

D3.3: Final report on the DESIRE6G intelligent and secure management, orchestration, and control

HEU 6G SNS JU Project

Grant No. 101096466



Co-funded by
the European Union

Document properties

Document number	D3.3
Document title	Final report on the DESIRE6G intelligent and secure management, orchestration, and control
Work Package	3
Editors	Anastassios Nanos (NUBIS)
Authors	Cyril Hsu, Chrysa Papagianni, Anestis Dalgkitsis , Marios Avgeris (UVA), Luis Velasco, Marc Ruiz, Jaume Comellas, Sima Barzegar, Davide Careglio, Josep Prat (UPC), Carlos J. Bernardos (UC3M), Vincent Lefebvre, Mark Agoustures (TSS), Alessandro Pacini, Luca Valcarenghi, Andrea Sgambelluri (SSSA), Anastassios Nanos, George Ntoutsos, Christos Panagiotou, Konstantinos Papazafeiropoulos, Maria Gkeka, Maria Goutha, Charalampos Mainas, Apostolos Giannousas (NUBIS), Chamith Mawela, Chathuranga Weeraddana (UOU), Sultan Ertaş, Merve Saimler (EBY), Michele Gucciardo (NEC), Rafael López, Miguel A. Carnero (TID)
Internal reviewers	1. Chathuranga Weeraddana (UOU), 2. Vincent Lefebvre, Mark Angoustures (TSS)
External reviewers	1. Chrysa Papagianni (UVA), 2. Gergely Pongrácz (ERI-HU)
Dissemination level	PU (Public)
Status of the document	Final version
Version	1.0
File name	D3.3_Final_report_intelligent_secure_management_orchestration_control_platform
Contractual delivery date	August 31 st , 2025
Delivery date	September 3 rd , 2025

Document history

Revision	Date	Issued by	Description
V0.1	20/03/2025	NUBIS	Initial ToC/Sections
V0.2	25/07/2025	NUBIS	Collection and consolidation of partners contribution
V0.3	04/08/2025	NUBIS	Released for internal review
V0.4a	18/08/2025	TSS	Internal review delivered
V0.4b	18/08/2025	OULU	Internal review delivered
V0.5	21/08/2025	NUBIS	Address review comments and release for external review
V0.6-8	26-31/8/2025	UVA, ERI-HU, NUBIS	Polish text, External Review.
V0.9	31/08/2025	EBY, TSS, UPC, SSSA, UVA, NUBIS, ERI-HU	Address external review comments & Polish text, Iterate over addressed comments.
V1.0	3/09/2025	NUBIS, UVA	Final text, ready for publishing.

Abstract

This document summarizes the work in WP3, presenting the latest and final developments of the architecture of the DESIRE6G Service Management and Orchestration Layer, along with its components and the technologies used to achieve flexible and intelligent service instantiation and management on a 6G network. This document is an extension of D3.1 [1] and D3.2 [2] with the implementation of the DESIRE6G secure management, orchestration, and control platform.

Keywords

SMO, MAS, Intent-based translation, DLT federation, Security, Policy

Disclaimer



This project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement No 101096466.



DESIRE6G is supported by the Smart Networks and Services Joint Undertaking.

This report reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

1. Introduction	16
1.1 Alignment of WP3 Components with System Architecture	18
1.2 Requirements alignment with objectives	19
1.2.1 E2E Service Orchestration Requirements.....	23
1.2.2 Autonomous Networking Requirements.....	23
1.2.3 Programmable Data Plane Requirements.....	24
1.2.4 AI Integration Requirements.....	24
1.2.5 Pervasive Monitoring Requirements.....	25
1.3 KPIs	26
1.4 AI/ML Enablement Across the Stack	31
2. E2E Service Management and Orchestration.....	32
2.1 DESIRE6G SMO.....	32
2.1.1 SMO Implementation.....	32
2.1.1.1 Service Orchestrator	33
2.1.1.2 Topology.....	35
2.1.1.3 Service Catalog	36
2.1.1.4 Data Lake.....	38
2.1.2 Internal and External Interfaces.....	39
2.1.2.1 Internal Interfaces.....	39
2.1.2.2 External Interfaces.....	40
2.1.2.3 Message Queue (RabbitMQ) – A Deeper Look.....	41
2.1.3 Challenges.....	42
2.1.4 SMO Deployment.....	44
2.1.5 Cloud-native deployment targets.....	45
2.2 Optimization Engine	47

2.2.1 AI/ML Workflow.....	47
2.2.2 Model Pool.....	48
2.2.3 Model Management and Selection.....	49
2.2.4 Optimization Subsystem Implementation.....	49
2.2.5 Encoding/Decoding and Processing Service Requests	50
2.3 IBN Management	52
2.3.1 SLA-to-Intent Translation.....	53
2.3.2 Implementation and Workflow of the Intent Management Function (IMF)	56
2.3.3 Intent Extension Models for Cross-Domain Automation in Digitalized Manufacturing.....	58
2.3.4 Autonomous Decision-Making Framework for Supporting Multiple Services in the IMF	62
2.4 DLT Federation	65
2.4.1 DLT-based Federation Implementation.....	65
2.4.2 DLT-based Federation Operation	68
2.5 Data lake synchronization	69
2.5.1 Reference scenario	70
2.5.2 Data synchronization.....	70
3. Service Optimization and Assurance	72
3.1 MAS Deployment	72
3.2 MAS-based Service Operation	75
3.3 MAS-based Service Reconfiguration.....	78
3.4 MAS Security	81
3.4.1 Security considerations of spontaneous attestation workflow.....	81
3.4.1.1 Problem statement.....	81
3.4.1.2 Agent-specific and one-time execution token mechanism	81
3.4.1.3 Practical implementation in D-MUTRA	83
3.4.2 Operational considerations. Fixing latency at start	85
3.4.2.1 Problem statement.....	85

3.4.2.2 Continuous attestation	85
3.4.2.3 Attest after starting security pattern	88
3.4.3 Novel drop-off zero-touch workflow	88
3.4.3.1 Problem statement	88
3.4.3.2 Novel security scheme	89
3.4.4 Blockchain considerations	90
3.4.4.1 From Ethereum PoS to hyperledger fabric	90
3.4.4.2 Limitation of Blob inflation rate	90
3.4.5 SECaaS hardening support to trusted execution	90
3.4.5.1 SGX implementation	90
3.4.5.2 SGX (partial) deprecation and replacement	93
3.4.5.3 SECaaS support of novel forms of TEEs	94
4. Robust, Sustainable and Privacy Preserving AI in DESIRE6G	98
4.1 Non-RT Decision Making at SMO	98
4.1.1 ML-Based Assisted SLA Decomposition	98
4.1.2 Transformer-Empowered Actor-Critic Service Function Chain Partitioning	102
4.1.3 Algorithmic Developments within OE	106
4.1.3.1 Model Pool Optimization Models	106
4.1.3.2 Model Selection	107
4.2 Near-RT Decision-Making at MAS	109
4.2.1 Autonomous Flow Routing and Subsequent nRT flow operation	109
4.2.1.1 MAS Configuration	110
4.2.1.2 Autonomous Flow Routing, Running in the Agents	112
4.2.1.3 Service Operation	113
4.2.2 Near RT Elastic Scaling Mechanisms	116
4.3 Complementary AI/ML Algorithms	120

4.3.1 Distributed Edge Intelligence for RIS.....	120
4.3.1.1 Implementation of algorithms.....	120
4.3.1.2 Implementation of algorithms within DESIRE6G architecture.....	122
4.3.2 Confidentiality Preserving Data Exchange with Third Party xApps	124
4.3.3 Forecasting Element to Support 6G Operations	129
4.3.4 Algorithms with Theoretical/Practical Insights	132
5. Regulatory Aspects for Proposed AI/ML Approaches.....	133
6. Conclusions	136
7. References	137

List of Figures

FIGURE 1 HIGH LEVEL ARCHITECTURE.....	18
FIGURE 2 MAPPING OF AI/ML COMPONENTS TO THE LAYERS OF THE ARCHITECTURE.	19
FIGURE 3: DESIRE6G SMO HIGH-LEVEL ARCHITECTURE.....	32
FIGURE 4: SERVICE ORCHESTRATOR REST API	34
FIGURE 5: TOPOLOGY REST API	35
FIGURE 6: SERVICE CATALOG REST API.....	37
FIGURE 7: DATA LAKE HIGH-LEVEL INTERFACES.....	38
FIGURE 8. ARCHITECTURE FOR ONBOARDING RESOURCE CONSTRAINED DEVICES	46
FIGURE 9: OPTIMIZATION ENGINE AI/ML ARCHITECTURE.....	48
FIGURE 10. OPTIMIZATION ENGINE WEB GUI.	50
FIGURE 11. SERVICE REQUEST.	51
FIGURE 12. ERROR MESSAGE DURING PARTITIONING.....	52
FIGURE 13. SLA-TO-INTENT TRANSLATION FRAMEWORK.....	53
FIGURE 14. IMF MICROSERVICE WORKFLOW WITHIN SMO ENVIRONMENT	57
FIGURE 15. INTENT-ORIENTED MANAGEMENT HIERARCHY IN INDUSTRIAL ENVIRONMENTS.....	59
FIGURE 16. INTENT MANAGEMENT FUNCTIONS.....	63
FIGURE 17. INTEGRATION OF DLT-BASED FEDERATION INTO THE DESIRE6G ARCHITECTURE.....	66
FIGURE 18. OPERATIONAL WORKFLOW OF DLT-BASED FEDERATION (EXAMPLE: SERVICE MIGRATION ACROSS MULTIPLE DOMAINS)	68
FIGURE 19. REFERENCE SCENARIO	70
FIGURE 20. DATA SYNCHRONIZATION APPROACH	71
FIGURE 21 DESIRE6G ARCHITECTURE FOR SECURE DISTRIBUTED INTELLIGENCE.	72
FIGURE 22 MAS PIPELINE DEPLOYMENT WITH INITIAL MUTUAL ATTESTATION.	74
FIGURE 23 EXAMPLE: DYNAMIC FLOW ROUTING.	76
FIGURE 24 NEAR-REAL-TIME OPERATION WORKFLOW.....	78
FIGURE 25 ILLUSTRATIVE MOBILITY SCENARIO (A) AND PROPOSED WORKFLOW (B).....	79
FIGURE 26. ON-DEMAND INTEGRITY MEASUREMENT IMPLEMENTATION.....	87
FIGURE 27 SECAAS-FREE DOCKER COMPOSE BASED WORKLOAD DROP-OFF WORKFLOW.....	89
FIGURE 28 . SYSTEMIC USER ACTIVATION OF SGX	92
FIGURE 29. PERFORMANCE PENALTY INDUCED BY SGX ON SYSTEMIC.....	93
FIGURE 30 SLA DECOMPOSITION.	99
FIGURE 31 ILLUSTRATION OF RADE FRAMEWORK.	100
FIGURE 32 AVERAGE E2E ACCEPTANCE PROBABILITY VS. ARRIVAL RATE.	101
FIGURE 33 AVERAGE E2E ACCEPTANCE PROBABILITY VS. CORRUPTION RATE AT ARRIVAL RATE=0.5.	101
FIGURE 34. OVERVIEW OF THE SDAC FRAMEWORK WITH TRANSFORMER-BASED ARCHITECTURE FOR SEQUENCE-AWARE SFC PARTITIONING.	103
FIGURE 35. ACCEPTANCE AND ARRIVAL RATE OVER TIME.....	105
FIGURE 36. AVERAGE ACCEPTANCE RATE.....	105
FIGURE 37: PARTITIONING SUCCESS RATE AND AVERAGE INFERENCE TIME PER MODEL.....	109
FIGURE 38 FLOW OPERATION UNDER TRAFFIC UNCERTAINTY.	111
FIGURE 39 FLOW OPERATION UNDER TRAFFIC UNCERTAINTY.	114
FIGURE 40 DETECTION OF BACKGROUND TRAFFIC MISCONFIGURATION.....	115
FIGURE 41 C-V2X SYSTEM OVERVIEW.....	117
FIGURE 42 DHGP FRAMEWORK.	118
FIGURE 43: AVERAGE REWARD OVER 40 DIFFERENT TRACES OF A 5.5H INTERVAL.	119

FIGURE 44 BEHAVIOUR OF ALL SOLUTIONS AS THE NUMBER OF VEHICLES CHANGES OVER TIME.	119
FIGURE 45 OVERVIEW OF FEDERATED LEARNING ARCHITECTURE INTEGRATION WITH 5TONIC (PHASE 4).	121
FIGURE 46 TRAINING ACCURACY AND TRAINING LOSS FOR CLASSIC FL AND FL GAMES WITH RIS DATA.	122
FIGURE 47 PROPOSED APPROACH FOR DATA EXCHANGE WITH A THIRD PARTY.	125
FIGURE 48 TESTBED SETUP.	126
FIGURE 49 EVALUATIONS FOR DIFFERENT ALGORITHMS WITH DIFFERENT SENARIOS.	129
FIGURE 50 FORECASTING ELEMENT ARCHITECTURE.	130
FIGURE 51 FORECASTING ELEMENT INPUT/OUTPUT LOGIC.	131
FIGURE 52. FORECASTING ELEMENT PERFORMANCE WITH K8S POD WITH (LEFT) AND WITHOUT (RIGHT) NUMBER OF EMULATED USERS.	132

List of Tables

TABLE 1 REQUIREMENTS AND COMPONENTS MAPPINGS.	20
TABLE 2. SLA-TO-INTENT TRANSLATION ALGORITHM	56
TABLE 3. SECURITY OF EXECUTION TOKEN COMPARISON.	82
TABLE 4 COMPARISON OF TECHNIQUES FOR CONTINUOUS ATTESTATION.	87
TABLE 5. ENUMERATION OF REQUIREMENTS TO EXPLOIT TDX AND SEV-SNP TEE	96
TABLE 6 NOTATION (MAS CONFIGURATION ALGORITHMS).	110
TABLE 7 ALGORITHM 1. ROUTE COMPUTATION ALGORITHM.	112
TABLE 8 OPTIMALITY GAP.	119
TABLE 9 EXAMPLE: ADOPTED DATASET.	125
TABLE 10 RESULTS FOR SCRAMBLED DATA.	127
TABLE 11 RESULTS FOR NON-SCRAMBLED DATA.	128

List of Snippets

SNIPPET 1: SMO REPOSITORY DIRECTORY STRUCTURE	44
SNIPPET 2: CONTAINER IMAGE BUILD	45
SNIPPET 3: DEPLOY THE SMO INTO AN EXISTING K8S CLUSTER.	45
SNIPPET 4: LOCAL DEPLOYMENT FOR DEBUGGING	45
SNIPPET 5: PARTITIONED PARTIAL SERVICE REQUEST YAML.	52

List of Acronyms

AWET	Absolute Weight Transformation
AMI	Adjusted Mutual Information
ARI	Adjusted Rand Index
AD	Administrative Domain
AF	Application Function
API	Application Programming Interface
AI	Artificial Intelligence
AAS	Asset Administration Shell
C-V2N	Cellular Vehicular-to-Network
CPU	Central Processing Unit
CAV	Connected Automated Vehicles
CCA	Confidential Compute Architecture
CNST	Constant
CSP	Content-Security-Policy
CP	Control Plane
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DHPG	Deep Hybrid Policy Gradient
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
DLT	Distributed Ledger Technology
E2E	End-to-End
ETSI	European Telecommunications Standards Institute
EU	European Union
FA	Failure Actuator
FEDAVG	Federated Averaging
FL	Federated Learning
FGOR	Fixed Gradient over Rays
GRU	Gated-Recurrent Unit
GDPR	General Data Protection Regulation
GUI	Graphical User Interface

Grpc	gRPC Remote Procedure Calls
INT	In-band Network Telemetry
I4.0	Industry 4.0
IML	Infrastructure Management Layer
IMF	Intent Management Function
IBN	Intent-based Network
IRM	Invariant Risk Minimization
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LLM	Large Language Model
LSTM	Long-Short Term-Memory
ML	Machine Learning
MLFO	Machine Learning Function Orchestrator
MANO	Management and Orchestration
MDP	Markov Decision Process
MLaaS	ML as a Service
MAS	Multiagent System
NLP	Natural Language Processing
nRT	near-RT
NF	Network Function
NFV	Network Function Virtualization
NS	Network Services
NN	Neural Network
OGD	Online Gradient Descent
O-RAN	Open Radio Access Network
OE	Optimization Engine
PS	Parameter Server
PoP	Points of Presence
PCA	Principal Component Analysis
PAA	Probability-as-Action
PDP	Programmable Data Plane
PI	Proportional and Integral
QoS	Quality of Service
RAN	Radio Access Network

RT	Real-Time
RADE	Real-time Adaptive DEcomposition
RX	Receive
RIS	Reflecting Intelligent Surface
RL	Reinforcement Learning
REST	Representational State Transfer
RDF	Resource Description Framework
RADG	Retrieval-Augmented Generation
RoT	Root of Trust
SDO	Standards Development Organizations
SECaaS	SECurity-as-a-Service
SEV-NP	Secure Encrypted Virtualization – Nested Paging
SLSQP	Sequential Least Squares Programming
SLO	Service Level Objective
SMO	Service Management and Orchestration
SO	Service Orchestrator
SLA	Service-Level Agreement
SP	Shortest Path
S3	Simple Storage Service
SC	Smart Contract
SDN	Software-Defined Networking
ToC	Table of Content
TX	Transmit
TCB	Trusted Computing Base
TCP	Transport Control Protocol
TDX	Trust Domain Extensions
TN	Transport Network
TES	Triple Exponential Smoothing
TEE	Trusted Execution Environment
TD3	Twin Delayed Deep Deterministic Policy Gradient
URLLC	Ultra-Reliable Low Latency Communications
V2I	Vehicle-to-Infrastructure
V2N	Vehicle-to-Network
V2P	Vehicle-to-Pedestrian

V2V	Vehicle-to-Vehicle
VXLAN	Virtual eXtensible Local Area Network
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function
WSS	Wavelength Selective Switch
WP	Work Package

Executive Summary

This deliverable, D3.3 – Final report on intelligent secure management, orchestration and control platform, presents the final architecture specification, component detailing, and design evolution of the DESIRE6G intelligent management and orchestration platform. It is the continuation and completion of the interim specification provided in D3.2 [2], now enriched by system-level feedback, cross-WP alignment, pilot input, and component-level advancements.

The DESIRE6G platform aims to enable a fully AI-native, secure, and intent-driven orchestration solution spanning across the Infrastructure, Network, and Service layers. This deliverable describes the finalized architectural principles, the latest iteration of the control and orchestration stack, and a detailed overview of the platform components, their roles, interactions, and integration trajectory.

This final version of the platform design has been completed in conjunction with the progress of WP4 and WP5 along with the architectural and security guidance from WP2. The deliverable also includes updated interaction diagrams, role-to-component mappings, and integration considerations that prepare the ground for the platform's technical validation and pilot deployment phases.

Overall, D3.3 concludes the design cycle of the DESIRE6G orchestration platform by delivering a comprehensive, modular, and deployable framework capable of supporting the demanding orchestration and control needs of future 6G networks with security, flexibility, and autonomy as its core principles.

1. Introduction

During the period M24-M32 of the project, the involved partners in WP3 have been working on consolidating the design and architectural work of the DESIRE6G management and orchestration platform initiated in D3.1 and continued in D3.2.

First of all, D3.3, aligns in Section 1 the architectural refinements with security requirements (D2.3) and system-level design inputs (D2.2), ensuring that the platform adheres to DESIRE6G's core design principles: zero trust, intent-based control, and AI-driven decision-making. This requirement mapping is followed by the project KPIs and how these have been reached in the context of WP3 activities.

The updated component descriptions, functional flows, and interaction diagrams in D3.3 demonstrate how the DESIRE6G platform translates architectural principles into concrete, implementable components that can be deployed in pilot environments (WP5). Notably, the platform embraces the notion of closed-loop orchestration, where telemetry feedback, AI inference, and policy adaptation cycles are integrated across all layers to deliver intelligent and proactive control.

D3.3 adopts a cross – WP alignment that reflects particularly with WP2 (Requirements & Architecture), WP4 (Unified Programmable Data Plane Layer), and WP5 (Integration, Validation & Demonstration). This deliverable elaborates the final version of the three-layer orchestration model, distinguishing among the Infrastructure, Network, and Service Orchestration Layers, each with clearly defined roles, capabilities, and communication interfaces.

D3.3 extends in Section 2 the Service Management and Orchestration (SMO) layer with a deployment strategy and automations for both Kubernetes and local deployments. Moreover, the deployment targets are now extended including Cortex-M class devices allowing new deployment models across energy efficient edge scenarios. As part of the E2E Service Management and Orchestration, D3.3 narrows down the optimization scenario that is subject of the service deployment and its partitioning optimizations according to SLAs and SLOs. In the same section, D3.3 enriches the Intent-Based Networking (IBN) management functions beyond the foundations laid in D3.2. While the previous version focused on SLA-to-intent translation, normalization, and initial workflows for service graph generation, D3.3 extends these capabilities with intent extension models for cross-domain automation in industrial manufacturing environments. Furthermore, an autonomous decision-making framework is introduced to support multi-service orchestration in 6G networks, enabling intents to directly drive closed-loop automation across heterogeneous services.

Moreover, Section 2, advances the DLT Federation capabilities initially introduced in D3.2. D3.3 emphasizes the operational aspects of federation by detailing end-to-end processes for service migration across multiple administrative domains. The updated design refines the implementation, shifting from conceptual scenarios to practical workflows, while ensuring trust anchors, SLA negotiations, and resource advertisements are executed securely.

Additionally, Section 2 introduces Data Lake synchronization, which was not explicitly addressed in D3.2. This addition ensures consistent data management across distributed sites, focusing on synchronization of telemetry, models, and service-related artifacts between local and central repositories. A reference scenario is defined to demonstrate how synchronization supports real-time analytics, federated AI workflows, and multi-domain orchestration.

In Section 3, D3.3 enhances the Service Optimization and Assurance framework by building on the MAS-centric architecture of D3.2. While the earlier deliverable focused on initial MAS deployment, service operation, and security through bootstrap authentication and attestation, D3.3 expands these capabilities with advanced reconfiguration mechanisms and continuous attestation workflows. Approaches are introduced to address latency reduction at startup, zero-touch security patterns, and blockchain-backed enhancements for trust management. Furthermore, SECaaS integration evolves to support diverse trusted execution environments beyond SGX, ensuring stronger resilience and adaptability.

D3.2 introduced ML-based SLA decomposition, reinforcement learning for MAS decision-making, and initial complementary algorithms for edge intelligence, privacy-preserving data exchange, forecasting, and federated learning. Building on these foundations, D3.3 incorporates in Section 4 transformer-based actor-critic models for service function chain partitioning, expands the Optimization Engine with refined model pool and selection processes, and strengthens MAS near-real-time decision-making with elastic scaling and flow routing improvements. Complementary AI algorithms are further matured through practical implementations in RIS-based edge intelligence, privacy-preserving xApps, and forecasting integrated within Kubernetes-driven testbeds.

Finally, the regulatory aspects for proposed AI/ML approaches in Section 5 are further expanded to take into consideration risk-based approaches (unacceptable risk, high risk, limited risk, minimal risk). The deliverable concludes in Section 6.

1.1 Alignment of WP3 Components with System Architecture

The DESIRE6G system architecture is depicted in Figure 1 and serves as a reference point throughout this document. The contributions of WP3 encompass the first two layers of the architecture (i.e., Intent-Based Orchestration layer, the Optimization and Network Control layer), as well as the AI integration throughout all layers. As such, this section presents the details of the mapping of AI/ML components to the architecture. The details of the mapping of design specifications across all layers of the architecture within the scope of WP3 are presented in this deliverable. This section outlines also the requirements established in D2.1 along with the corresponding components designed to address them.

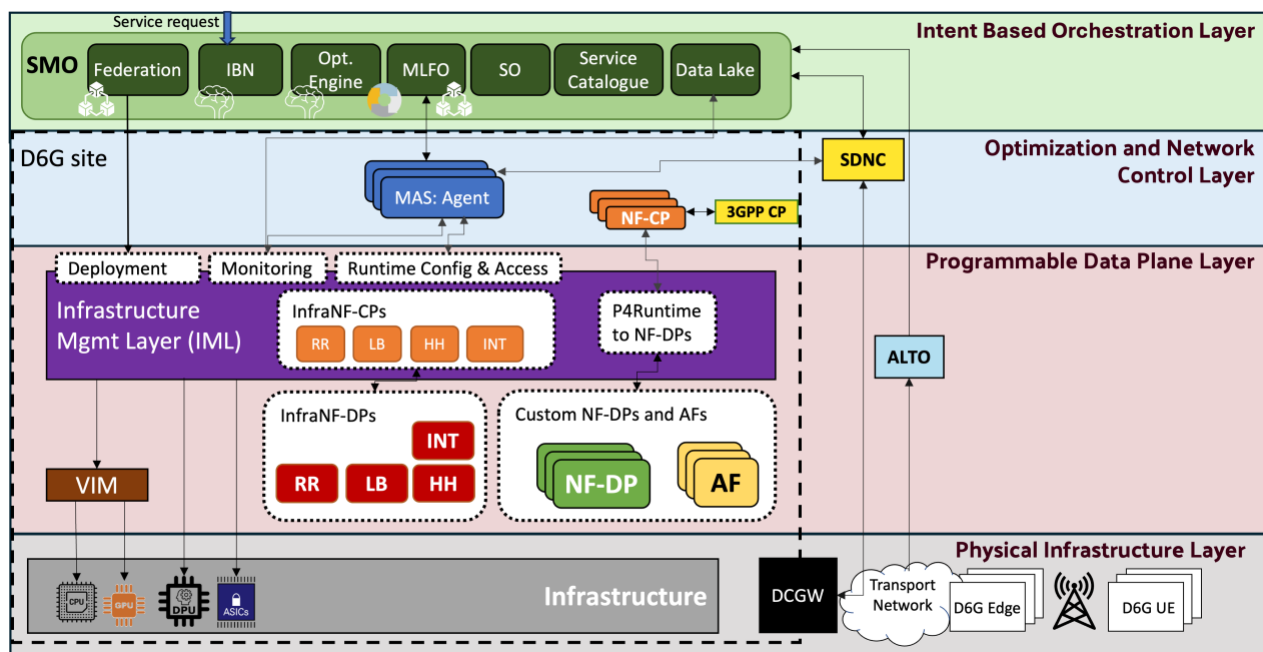


FIGURE 1 HIGH LEVEL ARCHITECTURE.

Propelled by the AI integration requirements, we start by carefully identifying and aligning suitable AI technologies to meet the design specifications pertaining to E2E Service Orchestration requirements, Autonomous Networking requirements, Programmable Data Plane requirements, and Pervasive Monitoring requirements at different layers of the architecture. As a result, the mapping of AI components to the DESIRE6G architecture is shown in Figure 2, together with the respective WPs within which the AI developments are performed. Note that some complementary AI tools are directly implemented in the Physical infrastructure layer as well. Details are summarized in the rest of this section.

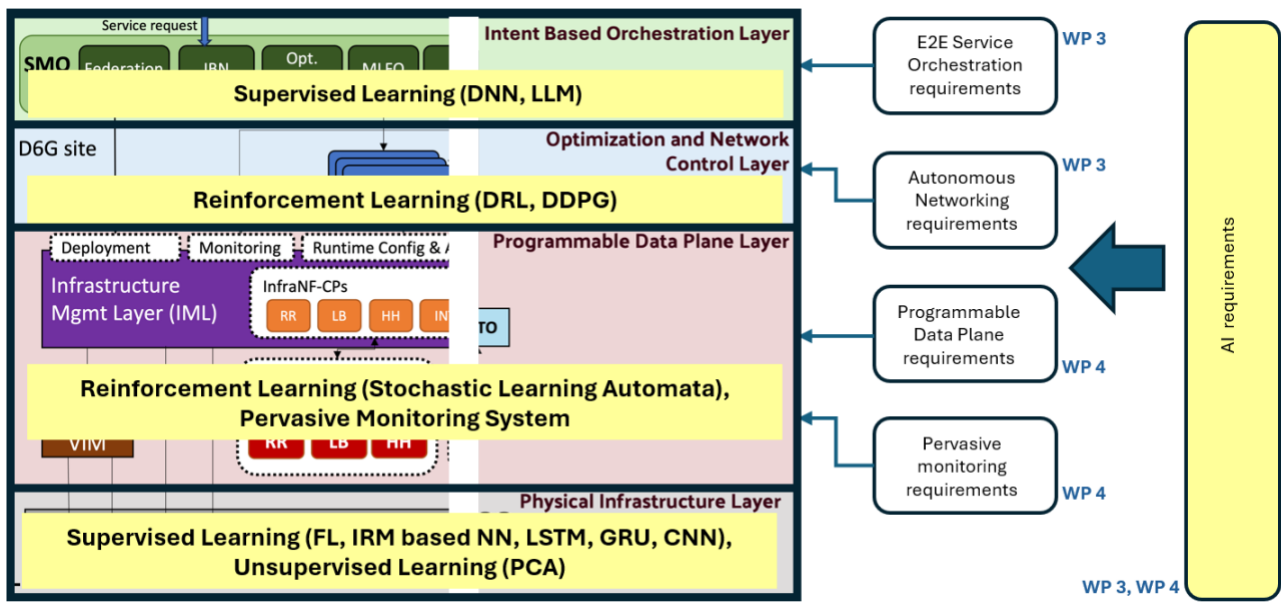


FIGURE 2 MAPPING OF AI/ML COMPONENTS TO THE LAYERS OF THE ARCHITECTURE.

1.2 Requirements alignment with objectives

This section provides a detailed mapping (Table 1) between the requirements as identified in WP2 and the architectural components and mechanisms introduced in the deliverables of WP3 (D3.1, D3.2 and D3.3). Each requirement is examined from the perspective of WP3, focusing exclusively on the technologies, orchestration capabilities, and intelligent control frameworks designed and implemented within this work package. The goal is to demonstrate how the introduced components contribute to fulfilling the project-wide design and performance objectives. The justification follows a per-requirement structure, elaborating on how each need is addressed through specific mechanisms, protocols, deployment strategies, or AI-driven capabilities. The technical details of each of the components are presented in more details later in the document.

TABLE 1 REQUIREMENTS AND COMPONENTS MAPPINGS

			Intent-based Orchestration Layer					Optimization and Network Control Layer						Programmable Data Plane Layer		
			Federation	IBN	Opt. Engine	MLFO	SO	MAS: Agent	SECaaS / D-MUTRA	NF-CP	Pervasive Monitoring	SDNC	IML	InfraNF-DPs	Custom NF-DPs and Afs	VIM
E2E Service Orchestration Requirements	E2E_SO-REQ-1	Network slicing					P									
	E2E_SO-REQ-2	Intent Based Management		P			P									
	E2E_SO-REQ-3	Cloud Native approach					P									
	E2E_SO-REQ-4	Federation of administrative domains	P				P									
Autonomous Networking Requirements	AN-REQ-1	Distributed decision-making			S			P					S			
	AN-REQ-2	Orchestration of AI/ML pipelines				P										
	AN-REQ-3	Security of MAS Agents						S	P							
	AN-REQ-4	Trust between agents						S	P							
	AN-REQ-5	Securing AI/ML pipelines and MAS inter-agent communications						S	P							

Programmable Data Plane Requirements	PDP-REQ-1	Unified Programmable Data Plane layer								S			P	S		
	PDP-REQ-2	Cloud Native approach								S			P			
	PDP-REQ-3	Slicing, Multitenancy and QoS enforcement								S			P			
	PDP-REQ-4	Offloading of network/service /AI functions										S	P	S		
	PDP-REQ-5	Integration of customised telemetry									S		P	S		
AI Integration Requirements	AI-REQ-1	AI models Integration in different architectural levels			P		P						P			
	AI-REQ-2	AI models to optimize the performance			P											
	AI-REQ-3	AI models data privacy			P											
	AI-REQ-4	AI models Sustainability			P											
	AI-REQ-5	Ai Algorithm robustness			P						S					

Pervasive Monitoring Requirements	PM-REQ-1	Conducting both service monitoring and infrastructure monitoring			S		S				P		P			
	PM-REQ-2	Pervasive monitoring system must extend all the infrastructure domains					S				P		P			
	PM-REQ-3	Pervasive monitoring system must extend to the UE									P				P	
	PM-REQ-4	Monitoring data to AI/ML data consumers in the architecture			S	S					P					
	PM-REQ-5	Pervasive monitoring system must be scalable									P					
	PM-REQ-6	Pervasive monitoring system must be customizable						S			P					

1.2.1 E2E Service Orchestration Requirements

E2E_SO-REQ-1: Network Slicing: Addressed by the Service Orchestrator (SO), which can manage end-to-end slice definitions, instantiations, and lifecycle operations across domains. The SO ensures the orchestration of vertical-specific service instances using slice templates and isolation constraints.

E2E_SO-REQ-2: Intent-Based Management: Implemented via the combination of the Intent-Based Networking (IBN) interface and the SO. The IBN module interprets high-level user intents and translates them into orchestratable policies and service requests, which are then realized by the SO across the stack.

E2E_SO-REQ-3: Cloud Native Approach: The SO is implemented as a microservice-based, containerized component, following cloud-native design principles. It supports dynamic scaling, stateless operations, and integration with CI/CD pipelines to align with cloud-native orchestration workflows.

E2E_SO-REQ-4: Federation of Administrative Domains: Fulfilled by the Blockchain Federation Manager and the SO, enabling multi-domain service orchestration across administrative boundaries. These components manage trust anchors, domain capabilities, SLA negotiations, and end-to-end orchestration across peers.

1.2.2 Autonomous Networking Requirements

AN-REQ-1: Distributed Decision-Making: Realized through the Optimization Engine, Multi-Agent System (MAS) Agents, and IML (Intelligent Management Layer). These elements decentralize orchestration decisions based on contextual data, local metrics, and inter-agent coordination.

AN-REQ-2: Orchestration of AI/ML Pipelines: Supported by the ML Function Orchestrator (MLFO), which provisions, manages, and scales AI/ML model deployments. The MLFO abstracts the underlying infrastructure, enabling lifecycle management of model pipelines as orchestratable resources.

AN-REQ-3: Security of MAS Agents: Addressed by embedding MAS Agents with security capabilities and integrating them with the SECaaS (Security-as-a-Service) and D-MUTRA (Multi-Trust Reasoning Agent) frameworks, ensuring agent authentication, secure communication, and policy compliance.

AN-REQ-4: Trust Between Agents: Achieved through the D-MUTRA and SECaaS modules, which enable agents to verify identities, exchange credentials, and enforce mutual trust policies before participating in orchestration decision loops.

AN-REQ-5: Securing AI/ML Pipelines and Inter-Agent Communications: Ensured through MAS Agent encryption hooks, secure telemetry ingestion, and D-MUTRA-managed policies that protect data exchanges and ML pipeline integrity, while preventing adversarial behaviors and model manipulation.

1.2.3 Programmable Data Plane Requirements

PDP-REQ-1: Unified Programmable Data Plane Layer: Enabled by the NF-CP (Network Function Control Plane), IML, and InfraNF-DPs (Infrastructure Network Function Data Plane nodes). Together, these components abstract data plane resources, enabling programmability and unified control.

PDP-REQ-2: Cloud Native Approach: Fulfilled by deploying NF-CP and IML components using containerized, stateless microservices that can be dynamically instantiated, monitored, and scaled within distributed edge/core cloud environments.

PDP-REQ-3: Slicing, Multitenancy, and QoS Enforcement: Implemented by the IML working in conjunction with the NF-CP, which applies isolation, per-slice traffic management, and SLA/QoS parameters at the data path level.

PDP-REQ-4: Offloading of Network/Service/AI Functions: Managed by SDNC (Software-Defined Network Controller), IML, and InfraNF-DPs, enabling function migration and runtime placement decisions to offload or move workloads closer to data sources or edge points.

PDP-REQ-5: Integration of Customized Telemetry: Supported by IML, InfraNF-DPs and Pervasive Monitoring, which together enable telemetry collection, filtering, and transformation tailored to different orchestration and ML subsystems.

1.2.4 AI Integration Requirements

AI-REQ-1: AI Models Integration Across Architectural Levels: Delivered through the Optimization Engine, SO, and IML, which jointly enable AI/ML modules at service, network, and infrastructure levels for cross-layer decision-making.

AI-REQ-2: AI Models to Optimize Performance: Executed by the Optimization Engine, which leverages historical and real-time data to drive policy adjustments, resource optimization, and anomaly prediction using reinforcement learning and Bayesian models.

AI-REQ-3: AI Model Data Privacy: Ensured by the Optimization Engine's privacy-preserving mechanisms, including model partitioning, selective telemetry ingestion, and secure data aggregation across domains.

AI-REQ-4: AI Model Sustainability: Addressed by incorporating energy-aware orchestration objectives into the Optimization Engine, which adjusts placement decisions and service activation based on energy efficiency metrics.

AI-REQ-5: AI Algorithm Robustness: Implemented through the combination of the Optimization Engine and Pervasive Monitoring, which validate model inputs, detect drift, and apply countermeasures in case of anomalous behavior or environment volatility.

1.2.5 Pervasive Monitoring Requirements

PM-REQ-1: Conducting Both Service and Infrastructure Monitoring: The platform addresses this requirement through the joint operation of the Optimization Engine, SO, Pervasive Monitoring system, and IML. The monitoring system observes both the physical infrastructure and virtualized services, enabling accurate cross-layer assessments. This enables the orchestrator to react based on insights such as SLA deviations, resource contention, or environmental events.

PM-REQ-2: Pervasive Monitoring System Must Extend to All Infrastructure Domains: Multi-domain observability is achieved through the deployment of monitoring agents across edge, RAN, core, and data center environments. The SO and IML coordinate telemetry streams, normalize metrics, and ensure alignment of monitoring objectives across all involved infrastructure segments. This guarantees full-stack visibility for orchestration and AI workflows.

PM-REQ-3: Pervasive Monitoring System Must Extend to the UE: The DESIRE6G platform extends monitoring capabilities to the User Equipment (UE) through lightweight telemetry endpoints. These are integrated with custom NF-DPs and application-layer functions, capturing performance and user-centric context directly from the edge. This enhances end-to-end observability and supports the delivery of personalized and adaptive services.

PM-REQ-4: Monitoring Data to AI/ML Data Consumers in the Architecture: A dedicated telemetry pipeline delivers structured, filtered, and context-aware monitoring data to AI/ML consumers such as the Optimization Engine and the MLFO. This enables real-time inference, training, and optimization

cycles driven by rich operational data. The interface supports both pull and push models, enabling flexible data delivery to various AI modules.

PM-REQ-5: Pervasive Monitoring System Must Be Scalable: The monitoring framework is designed to operate in cloud-native environments, with scalable microservice-based agents that adapt to system load and infrastructure complexity. Data aggregation is distributed and hierarchical, supporting performance in large-scale deployments without compromising latency or completeness.

PM-REQ-6: Pervasive Monitoring System Must Be Customizable: The monitoring system supports dynamic reconfiguration, allowing orchestration policies and MAS agents to define what, when, and how to monitor. This includes selective probe activation, tailored granularity, and domain-specific telemetry logic. Such customization ensures that only relevant data is collected, reducing overhead and improving effectiveness across diverse services and environments.

1.3 KPIs

Objective 1 (O1)	Design a functional architecture for 6G mobile networks to support the next generation of URLLC use cases
-------------------------	--

KPI 1.1 Meeting the evolving KPIs and forming the KVs for the DESIRE6G use cases that fall under extreme URLLC category

Status	Comment
Fulfilled	<ul style="list-style-type: none"> multi-level architecture, consisting of an E2E SMO and D6G sites neural network-based approach for SLA decomposition

Objective 2 (O2)	Employ a cloud-native approach to vertical service and mobile network deployments over heterogenous and dynamic resources that span across multiple administrative domains
-------------------------	---

KPI 2.1: Package and deploy in the continuum applications consisting of at least 10 components, transparently and on-demand targeting 4 different execution enclaves (VMs, microVMs, unikernels, containers), at comparable latency with state-of-the-art standalone enclave generation methods.

Status	Comment
Fulfilled	WP3 develops a pure cloud-native framework, able to spawn a workload packaged as an OCI artifact (container image) in various execution modes:

container, sandboxed container (microVM or VM) or unikernels. This includes: the involvement of NUBIS in the development of sandboxed container runtimes (kata-containers), the development of the packaging software (bima/bunny) and the container runtime for unikernels (urunc). Initial design & implementation of these features has been included in D3.1, MS3.1 and MS3.2, and further developments are reported in MS3.3 and D3.2/D3.3. Specifically, a service graph containing 4 components has already been demonstrated. Further enhancements to include more network functions are included as part of the 2nd reporting period, with a mixed deployment mode containing various runtime classes (generic containers, sandboxed containers and unikernels).

KPI 2.2: Support ARM Cortex-M family devices as legitimate targets for application deployment and resource orchestration

Status	Comment
Fulfilled	A cloud-native framework (compatible with k8s) has been developed in WP3, that is able to manage microcontroller-based devices via OCI (container) artifacts, in a pure cloud-native way (using the k8s API).

KPI 2.3: Federate two (2) experimental sites of the DESIRE6G demonstrators targeting different procedures involved in federation such as domain registration, advertisement/discovery of services and resources, negotiation, and deployment

Status	Comment
Fulfilled	Designed the DLT infrastructure, Smart Contract, APIs, and workflow for a DLT-based federation process within the DESIRE6G architecture as described in D3.1. Integrated and validated the initial and final versions of the DLT federation component in the 5TONIC testbed.

Objective 3 (O3) Design a “AI-native architecture” for 6G systems. While 5G solutions aimed at providing machine learning solutions over-the-top, DESIRE6G seeks to update the network architecture so that it natively supports AI operations.

KPI 3.1: Autonomous operation based on multi-agent systems to reduce > 25% OpEx w.r.t. manual/static operation.

Status	Comment
--------	---------

Fulfilled	<p>In the case of near-RT decision-making with distributed knowledge, the main component in the architecture is the MAS. Solutions based on deep reinforcement learning and multi-agent systems to create a distributed collaborative network control plane are considered, which bring intelligence closer to the data plane enabling reduced response times. As a result, up to sub-second granularity is achieved with autonomous decision-making based on agents' own (local) observed data, as well as on the data and models received from other agents (remote).</p> <p>The performance in terms of OpEx reduction of MAS in charge of the autonomous control of traffic flows supporting 6G connectivity services with stringent requirements has been numerically evaluated in [3]. MAS enables near-real time adaptation of resources to support connectivity services through multiple paths, thus achieving power savings larger than 30% with respect to the benchmarking approach, where connectivity is statically managed. Moreover, energy efficiency (actual flow traffic per power consumption unit) is increased between 50% and 100% for different 6G network scenarios.</p>
-----------	---

KPI 3.2: Near-real time local (inside a node) control loops, including data collection, analysis, and decision making in < 10 ms.

Status	Comment
Fulfilled	<p>Design and implementation of a generic MAS agent, which internal architecture consists of five main components: i) a manager module configuring and supervising the operation of the rest of the modules; ii) a security manager in charge of security aspects, like key management; iii) a number of algorithms for data collection, analysis, and decision making; iv) a number of interfaces to communicate with other systems; and v) a Redis DB that is used in publish-subscribe mode to communicate the different modules among them. This facilitates the definition of specific workflows for telemetry and control loops and provides an agile, reliable, and secure environment that simplifies communication, as well as integration of new modules.</p> <p>Near-real time operation control loop below 10 ms was experimentally validated in [4]</p>

KPI 3.3: <1s (depending on the involved elements) automatic resource selection and network programmability to validate the required workflows entailing the orchestration and control functions, and the defined interfaces

Status	Comment
Fulfilled	<p>Intelligence at edge is embedded as integral components in forecasting elements (FEs), confidentiality preserving data exchange mechanisms, resource management in a heterogeneous multi-reflecting intelligent surface (RIS) setting, and data-driven scaling mechanisms.</p> <p>The D6G SMO is designed based on a micro-service architecture. This brings fast response times between components and flexible control data exchange across the whole continuum.</p>

KPI 3.6: 95% accuracy and inference reliability coming from properly defined data sets

Status	Comment
Fulfilled	<p>Certain parts of the SMO developments are based on ML algorithms, such as the initial SLA decomposition machinery, which maintain accuracy levels at 95% by using well-defined datasets that meaningfully represent underlying data distributions. These results are documented in D3.1.</p>

Objective 6 (O6)	Develop a cross-domain, infrastructure-independent, software security by executable rewriting technology enabling trustworthy immersive process monitoring and remote control backed on lightweight permissioned Distributed Ledger Technology
-------------------------	---

KPI 6.1: Completeness of the security functions including self-authentication, software confidentiality and run-time verification, remote execution monitoring and control leveraging DLT overlay

Status	Comment
Fulfilled	<p>Authentication: A significant progress is taken towards the original self-authentication as stated above. We are now considering Remote attestation processed in a mutual arrangement where any (SECaaS-prepared) software can be used to remotely authenticate others. This results from our original study phase which reconsidered how DLT can be best used. Our DLT-based Mutual Remote Attestation (aka D-MUTRA) described in task T3.2 is a disruptive model for remote attestation.</p> <p>Confidentiality: Our integration work with UPC has confirmed that confidentiality preservation can be achieved by SECaaS text section</p>

	encryption produced on Cython. (native compiled Python programmed models through Cython rewriting).
--	---

KPI 6.2: Completeness of payload deployment contexts (i.e., presence or absence of Intel's SGX) for physical and cloud native containerized and virtualized deployments

Status	Comment
Fulfilled	Our work covers both native implementation and containerized payloads. We are currently working on a containerized sidecar layout which will be further integrated into our binary native wrapping SECaaS. We are doing that without further consideration to SGX, as being deprecated by Intel. Deliverable D3.3 pinpoints our support regarding to the payload type and the execution environment and notably VM based TEE (e.g., Intel's TDX).

KPI 6.3: 100% of X-86 platform ELF formatted binaries (including P4 compiled binaries for software targets).

Status	Comment
Fulfilled	Our work covers both native implementation and containerized payloads. We developed a containerized sidecar layout which is further integrated into our binary native wrapping SECaaS. The objective is to reach 100% x-86 native compiled coverage irrespective of the programming language. This deliverable D3.3 pinpoints our support regarding to various forms of x86 native compiled binaries from different sources (programming languages including P4)

KPI 6.4: 20% maximum average runtime overhead incurred by the composite security compared to non-protected versions of representative sample of each targeted software type as defined in KPI6.3 above. 0% increase of the one-time authentication complete cycle of the self-contained authentication novel scheme compared to TPM based authentication (given in the range of 500 msec for reference).

Status	Comment
Fulfilled	D-MUTRA has developed a novel mutual remote attestation scheme leveraging the blockchain, a significant step taken against our original self-authentication method in terms of security. Our "Attest After Starting" novel pattern enables to drop workload latency at start to nil, a strong benefit when compared with TPM based remote attestation. This scheme removes all blockchain based extra timing (which could

degrade the remote attestation cycle compared to TPM based remote attestation). Our Attest before starting remote attestation cycle including the blockchain processing is lower than 500 msec in our different tests.

D-MUTRA's runtime integrity verification has been deeply improved to reduce the performance penalty to less than 1%, leveraging our own spread-over-time measurement technique and Linux's cgroups resource management utility.

Last, our code confidentiality leveraging encryption does not impact performance as the original code runs and induces a penalty at start in the range of 10-50 msec.

1.4 AI/ML Enablement Across the Stack

AI/ML capabilities in DESIRE6G are embedded across all layers of the orchestration stack to enable intelligent, context-aware, and autonomous network and service management. At the service orchestration level, AI techniques support intent interpretation, SLA decomposition, and policy refinement. In the network control and optimization layer, machine learning models are leveraged for traffic prediction, resource allocation, and slice lifecycle management. At the infrastructure layer, AI components contribute to fine-grained telemetry analysis, anomaly detection, and performance forecasting. Together, these enablers form the foundation for closed-loop automation, allowing the system to monitor, decide, and adapt dynamically to evolving service requirements and network conditions. The architectural rationale and the conceptual mapping of these AI/ML functions across the orchestration layers were presented in detail in Section 2.3 of Deliverable D3.2.

2. E2E Service Management and Orchestration

One of the key objectives of WP3 is to design and implement a highly efficient End-to-End Service Orchestration system within the DESIRE6G framework. The focus is to enable seamless deployment and lifecycle management of services across a heterogeneous, multi-domain 6G environment. One of the key components to achieve this is the SMO. Related opensource developments and implementations are available at [5] [6] [7] [8] [9] [10] [11] [12] [13] [14].

Additional components include the secure MAS, along with AI/ML algorithms that facilitate service instantiation and re-configuration. These elements are essential for achieving secure, scalable, and autonomous service orchestration in DESIRE6G's cloud-native architecture.

2.1 DESIRE6G SMO

The design of the DESIRE6G SMO is described in D3.1 [1] and D2.2 [15]. This section provides additional details regarding the implementation of the SMO, along with the challenges related to both design choices and implementation limitations.

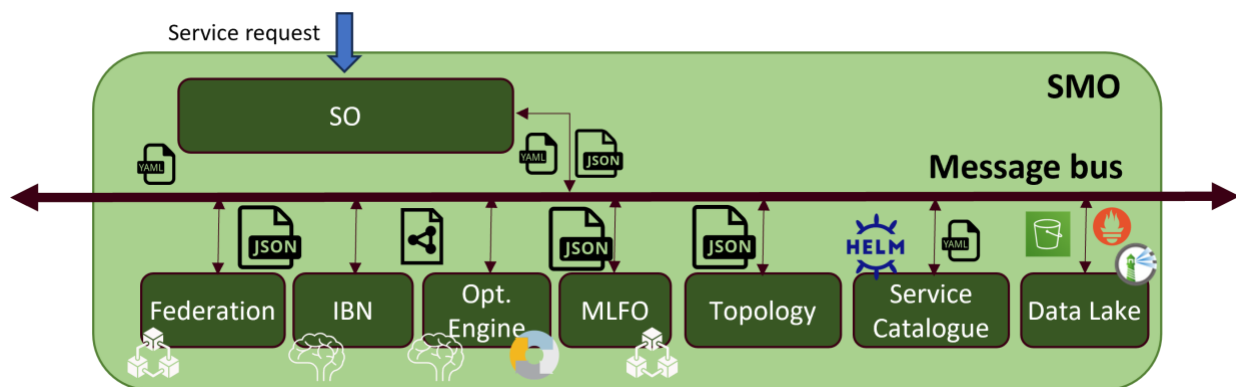


FIGURE 3: DESIRE6G SMO HIGH-LEVEL ARCHITECTURE

2.1.1 SMO Implementation

The implementation of the SMO leverages a modular, microservice-based architecture, to enable scalability and easy management (Figure 3). This design allows each component of the SMO to operate independently, providing flexibility for updates, scalability, and independent deployment. A key feature of this architecture is the integration of sandboxed container runtimes, such as kata-containers, which

ensure secure multi-tenancy. This approach aligns with cloud-native principles, supporting both workload isolation and security on shared infrastructure. In Task 3.1, significant advancements have been made to implement this architecture. The key components of the SMO, along with their implementations are described in the reset of this section.

2.1.1.1 Service Orchestrator

The Service Orchestrator coordinates and manages the lifecycle of services, handling tasks such as deployment, deletion, and listing of active services. It enables secure and modular service management, coordinating with the DESIRE6G sites (e.g., through IML to instantiate or terminate services as needed.

FastAPI [16] provides a RESTful [17] interface for creating, retrieving, and deleting service instances (Figure 4). The service orchestrator (SO) integrates with RabbitMQ [18] to receive and send messages to the bus, related to service status and updates, interacting with the rest of the SMO components. This component enables the instantiation of services and provides options to retrieve lists of all deployed services, as well as specific details for each one.

Service Orchestrator API1.0.0OAS 3.0

API for managing service deployments and tracking requests

servicesService management endpoints

POST

/services

Deploy a new service

GET

/services

List all deployed services

GET

/services/{service_id}

Get service details by ID

DELETE

/services/{service_id}

Delete a deployed service

requestsRequest management endpoints

GET

/requests

List all requests

GET

/requests/{request_id}

Get request details by ID

Schemas

ServiceRequest

name*

site_id*

> [...]

> [...]

FIGURE 4: SERVICE ORCHESTRATOR REST API

Related application programming interface (API) Endpoints are listed below with a brief description:

POST /services: Deploys a new service by ID.

Request Body JavaScript Object Notation (JSON):

```
{  
  "service_id": "string"  
}
```

Description: Deploys a service, based on the provided ID.

DELETE /services/{service_id}: Deletes a deployed service.

Description: Removes a specific service instance based on its unique `service_id`.

GET /services: Lists all currently deployed services.

Response (JSON):

```
[  
  {  
    "service_id": "string",  
    "name": "string",  
    "status": "string"  
  },  
  ...  
]
```

Description: Provides a summary of all active services.

GET /services/{service_id}: Retrieves details of a specific deployed service by `service_id`.

```
{  
  "service_id": "string",  
  "name": "string",  
  "status": "string",  
  "details": {...}  
}
```

Description: Returns information about a specific deployed service.

2.1.1.2 Topology

The Topology component is responsible for maintaining and updating the network and service topology, which includes tracking nodes (DESIRE6G sites) and their resources. It allows for the addition and retrieval of nodes, providing essential data for deployment and resource management decisions.

The component uses FastAPI to expose a RESTful API, which supports easy integration with other services (Figure 5). Data about nodes, such as central processing unit (CPU), memory, and storage, is stored in a database, which the Topology component can query and update. A RabbitMQ interface facilitates asynchronous notifications to other services, informing them of any updates to the topology.

Topology Manager API 1.0.0 OAS 3.0

API for managing nodes and links in the topology

nodes
Operations with nodes in the topology

POST
/nodes/
Add a node

GET
/nodes/
Get all nodes

GET
/nodes/{site_id}
Get a node by site_id

DELETE
/nodes/{site_id}
Delete a node

links
Operations with links in the topology

POST
/links/
Add a link between two nodes

GET
/links/
Get all links

Schemas

Node
{
 site_id* > [...]
 cpu* > [...]
 mem* > [...]
 storage* > [...]
 iml_endpoint* > [...]
}

Link
{
 source* > [...]
 destination* > [...]
 latency_ms > [...]
}

FIGURE 5: TOPOLOGY REST API

Related API Endpoints follow:

POST /nodes/: Adds a new node to the database.

Request Body (JSON):

```
{
  "site_id": "string",
  "cpu": "string",
  "memory": "string",
  "storage": "string",
  "iml_endpoint": "string"
}
```

Description: Stores node data, including the site ID, the IML endpoint and resource details.

GET /nodes/{site_id}: Retrieves node information by site_id.

Response (JSON):

```
{
  "site_id": "string",
  "cpu": "string",
  "memory": "string",
  "storage": "string",
  "iml_endpoint": "string"
}
```

Description: Retrieve information about a specific node, based on its ID.

2.1.1.3 Service Catalog

The Service Catalog acts as the registry for available services within the D6G platform. It stores service configurations, primarily in YAML format, which define how services should be deployed and configured. This catalog enables easy discovery and management of services within the architecture.

Using FastAPI, this component offers endpoints for storing and retrieving service definitions. Service configurations are stored in YAML format, making them easy to manage and version. RabbitMQ provides asynchronous notifications for updates, enabling real-time catalog synchronization.

Desire6G Service Catalog API 1.0.0 OAS 3.0

API for uploading, retrieving, listing, and deleting service graph and network function files.

catalog Desire6G Service Catalog component ^

GET	/catalog/{file_type}	List catalog files by type	▼
POST	/catalog/	Upload a file to the catalog	▼
GET	/retrieve/{file_name}	Retrieve a file from the catalog	▼
DELETE	/catalog/{file_name}	Delete a file from the catalog	▼

Schemas ^

FileType ▼ string
example: SERVICE_GRAPH
Enum:

☒ SERVICE_GRAPH, NETWORK_FUNCTION]

FIGURE 6: SERVICE CATALOG REST API

Related API Endpoints are listed below:

POST /catalog: Stores a YAML graph in the catalog

Request (File Upload): The uploaded file must have a .sg.yaml or .nf.yaml extension.

Response (JSON):

```
{
  "name": "string",
}
```

Description: Saves a service definition or configuration under a specific name.

Example CURL Command:

```
curl -X POST -F "file=@/path/to/service_graph.sg.yaml" http://localhost:8000/catalog/
```

GET /retrieve/{file_name}: Retrieves the contents of a YAML file by its **file_name**.

Example CURL Command:

```
curl http://localhost:8000/retrieve/example_file.yaml
```

Description: Returns the content of the specified YAML file.

GET /catalog/{file_type}: Lists all YAML files currently stored in the server's storage by file_type.

Example CURL Command:

```
curl http://localhost:8000/catalog/service_graph
```

Description: Provides a list of available YAML files, allowing clients to see all stored service configurations

DELETE /catalog/{file_name}: Delete the YAML definition identified by **file_name**.

Example CURL Command:

```
curl -X DELETE http://localhost:8000/catalog/example_file.yaml
```

Description: Returns 200 when the file is deleted from the catalog or error otherwise..

2.1.1.4 Data Lake

The Data Lake is a versatile repository structured to support the specialized data needs of its cloud and edge. It is optimized to store and manage both large datasets and operational insights, centralizing the information flow to support enhanced analytics, ML, and system-wide optimization.

S3 Storage is dedicated to housing ML models and large data blobs, ensuring scalable, high-performance storage for sizable, static assets. This approach allows the DESIRE6G platform to store and manage data that does not require real-time processing but is essential for training, retraining, and deploying ML models across the platform. By storing models centrally, the DESIRE6G system can deploy and update ML capabilities to nodes at all levels of the continuum as new models become available, ensuring adaptive, data-driven intelligence throughout the network.

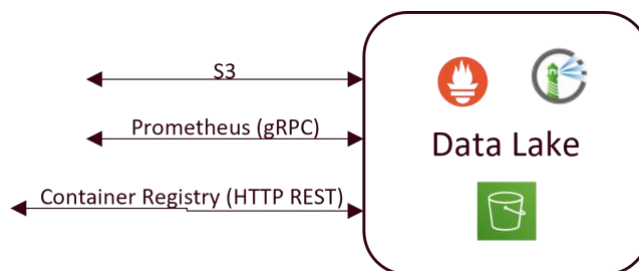


FIGURE 7: DATA LAKE HIGH-LEVEL INTERFACES

Telemetry ingestion is managed through a separate, high-frequency streaming endpoint using Prometheus and Kafka. This endpoint is responsible for real-time telemetry data, relaying telemetry from all nodes, including CPU and memory availability, usage, and latency. Prometheus manages time-series data, providing a historical record of system performance, while Kafka handles the high-throughput

streaming necessary for event-based data, facilitating continuous, near-instantaneous updates, and replication across all sites. This structured telemetry layer feeds analytics engines and real-time AI models within the DESIRE6G continuum, enabling closed-loop decision-making.

The Container Image Registry endpoint acts as a dedicated endpoint within the Data Lake for storing and managing container images. This repository is essential for providing service, network and application functions that are to be deployed in DESIRE6G. With this approach, all nodes have entire access to up-to-date software and services.

Alongside the components mentioned above, the SMO consists of the OE (Section 2.2) and the IBN (Section 2.3) described in their respective sections, fully integrated as micro-services in the overall SMO architecture.

2.1.2 Internal and External Interfaces

The SMO is built with both internal and external interfaces, each facilitating specific types of communication and data flow within and outside the SMO environment.

2.1.2.1 Internal Interfaces

Internal interfaces handle communication between SMO components. They enable the flow of information within the SMO, allowing each microservice to function effectively while contributing to the system's overall goals.

Message Queue (RabbitMQ):

The message queue is the backbone of internal communication within the SMO, providing asynchronous communication between components. This setup is essential for a distributed microservices architecture, as it decouples services, allowing them to communicate without needing to be aware of each other's implementation specifics.

Both the IBN component and the Optimization Engine rely on real-time data to perform optimizations and enforce intents. The message queue enables these components to receive up-to-date information about the network state, topology changes, service deployments, and performance metrics from other parts of the SMO. For instance, the Optimization Engine can subscribe to performance and resource data, analyzing it for optimization decisions.

The message queue supports both synchronous and asynchronous communication, which is crucial for handling tasks with different latency and criticality requirements. It manages events like service state updates, alerts, and topology changes, propagating them efficiently.

Data Model:

A shared data model standardizes the format for information exchanged between services, including JSON and YAML for configurations, metrics, and telemetry. Consistent data formats reduce parsing errors and facilitate smooth data exchange, allowing services like the Data Lake to store and retrieve information accurately. Each service exposes REST API endpoints (using FastAPI), which other services within the SMO can call. These endpoints allow services like the Service Orchestrator, Topology Manager, and Service Catalog to interact directly for operations that require immediate confirmation or response, such as retrieving node data or deploying new services.

Sync/Async Communication:

The SMO uses synchronous communication for requests that need immediate responses (e.g., querying a specific node's details) and asynchronous communication for tasks that can be processed in the background (e.g., service deployment). This hybrid communication model provides flexibility, ensuring that essential operations are handled promptly while less critical tasks do not overload the system.

2.1.2.2 External Interfaces

External interfaces expose SMO functionalities to external entities, such as other network components, third-party applications, or users.

REST APIs:

REST APIs provide accessible endpoints for external systems to interact with SMO functionalities, such as retrieving network topology, managing service deployments, and accessing stored data in the Data Lake. External systems (e.g., OSS/BSS logic) can submit high-level intents via REST, which the SMO then translates into actionable tasks. External monitoring tools or control systems can request performance reports or optimization suggestions via the REST API.

Simple Storage Service (S3)-Compatible Storage:

For components that handle large datasets such as logs, configuration files, or telemetry data) the SMO might provide an S3-compatible interface for easy, scalable storage and retrieval. The Data Lake, for instance, uses an S3 interface to store and retrieve telemetry data generated by network devices or application performance logs.

Time-Series Data and Remote Procedure Calls (gRPC):

gRPC-based interfaces allow external systems to perform high-speed data retrievals for monitoring, logging, or analytics. Time-series data is essential for performance monitoring, and gRPC enables efficient real-time data streaming, which could be particularly useful for the Network Optimization Engine in detecting trends and adjusting configurations accordingly.

2.1.2.3 Message Queue (RabbitMQ) – A Deeper Look

The message queue's role extends beyond simple message passing. In the SMO, RabbitMQ is configured to handle different types of messages (topics) and allow specific components to subscribe or publish as needed.

The message queue enables the SMO to adopt an event-driven paradigm, where components react to specific events (e.g., node addition, service deployment) in nRT. This is particularly useful for the IBN, which translates intents into network configurations, and the Network Optimization Engine, which relies on up-to-date data for effective optimization.

RabbitMQ allows each service to operate independently while still receiving updates. This decoupling is essential in microservices, ensuring that services like the Topology Manager or Service Orchestrator can update or deploy new services without interrupting other components.

RabbitMQ helps distribute workload among multiple instances of the same service, such as several instances of the Data Lake processing data storage and retrieval tasks. It improves the scalability and resilience of the SMO by distributing tasks evenly and handling high traffic during peak loads.

If a service fails, messages can be retained in queues until the service is back online. This guarantees that critical tasks (e.g., intent updates in IBN or optimization tasks) are not lost and are processed as soon as the service is available.

2.1.3 Challenges

In a microservices-based SMO, managing the complexity of interfaces poses a significant challenge. Each microservice may have unique requirements for data formatting, communication protocols, and request handling, leading to intricate and interdependent interfaces. This complexity becomes even more difficult to manage as the SMO scales, as services increasingly rely on coordinated communication to function correctly. Without the proper interface design and thorough testing, the system may suffer from data inconsistencies, delayed responses, or outright communication failures, all of which could impact the stability of the SMO. For instance, if the Topology Manager and Service Orchestrator services expect slightly different data formats, a mismatch could result in misinterpreted data, incorrect topology updates, or failed service deployments.

The scattered management that comes with a microservices-architecture is another challenge. Each service is deployed, monitored, and scaled independently, which adds flexibility but complicates centralized oversight and coordination. The need for independent lifecycle management for each service, from updates and monitoring to scaling and error handling, often leads to increased operational overhead. This can create challenges in troubleshooting and debugging, as tracking down the origin of issues may require insights from multiple services across the SMO.

Asynchronous communication, while enabling real-time, non-blocking interactions, also introduces potential delays and complexities in ensuring message consistency. Some operations depend on a series of events from other services, and without strict synchronization, delays in one service can cascade into delayed responses or missing data in others. For instance, if the Service Orchestrator depends on updates from the Topology Manager but receives them with a lag, it may inadvertently act on outdated data, affecting deployment accuracy.

The SMO must also address the demands of a multi-domain environment, where each component or module may involve different programming languages, data formats, and deployment environments. This diversity requires interoperability solutions to bridge these gaps, adding complexity to both development and maintenance. Multi-domain challenges are particularly pronounced in telecom settings, where compliance and performance requirements often differ across network domains. For example, the Network Optimization Engine might rely on data formats that differ from those used by

the Intent-Based Networking component, requiring translation or mapping mechanisms that add complexity and potential points of failure.

Together, these challenges highlight the complexity and need for well-coordinated, robust design in a microservices-based SMO, emphasizing the importance of clear interfaces, strict synchronization, and careful attention to interoperability across all SMO components. To address the challenges and limitations inherent in a microservices-based SMO, several mitigation techniques can be employed. These techniques aim to enhance the system's robustness, improve efficiency, and ensure the seamless operation of services while maintaining flexibility and scalability.

To mitigate the complexity of managing interfaces between microservices, one of the most effective strategies is to define clear API contracts. This means establishing well-documented, standardized formats for data exchanges between services. By using open standards like RESTful APIs and JSON or YAML for data interchange, we reduce mismatches and ensure smooth communication.

Moreover, employing automated integration tests can prevent interface issues before they arise in production. These tests can verify that each service can communicate as expected with other services, catching problems early in the development cycle.

To address the challenges posed by scattered service management, we employ centralized service orchestration. This involves using Kubernetes which manages the lifecycle of services across the system. Kubernetes automates deployment, scaling, and operation of containerized services, simplifying management and allowing services to be updated or replaced without impacting the whole system. For monitoring and alerting, we use Prometheus and Grafana, that provide real-time insights into the health and performance of individual services, helping us identify and resolve issues quickly.

Furthermore, centralized logging to the Data Lake, allows us to trace requests and identify failures in such a complex distributed system. This helps mitigate the difficulties of troubleshooting in a scattered microservices environment.

Mitigating delays and inconsistencies in asynchronous communication is achieved through message queue management strategies. Using RabbitMQ in combination with message acknowledgements ensures that messages are reliably delivered and processed, reducing the risk of data loss or out-of-order processing. Additionally, dead-letter queues capture messages that fail to be processed, enabling a mechanism for recovery without loss of critical data.

To further reduce potential delays, service prioritization is implemented. We define priority levels for different types of communication, ensuring that critical operations (such as service deployment or intent updates) are processed more urgently than less time-sensitive tasks. Event-driven architecture is also employed to ensure that the system only processes events that are relevant to the current context, avoiding unnecessary load on the system.

Dealing with multiple domains, languages, and deployment environments requires careful design of interoperability layers. Using middleware services that translate and map data between different formats and protocols ensures smooth communication between heterogeneous systems. In cases where different microservices use various technologies (e.g., Python, Go, Java), leveraging common communication standards such as gRPC allows disparate systems to communicate more effectively.

Additionally, containerization ensures that each service runs in a consistent environment, regardless of the underlying infrastructure, and tools like Helm help us package and deploy services in a standardized manner. This reduces friction in cross-domain interactions by providing consistent deployment strategies. For systems that rely on real-time, high-throughput communication like our SMOs, ensuring scalability and reliability is crucial. A combination of auto-scaling using tools like Kubernetes HPA and load balancing ensures that the system can dynamically adjust to varying loads. Kubernetes can automatically scale services based on demand, ensuring that the system remains responsive even as workloads fluctuate. Similarly, retry mechanisms are introduced to handle transient failures and ensure eventual consistency in communication.

2.1.4 SMO Deployment

The SMO (Service Management Orchestrator) repository is a key component of the Desire6G framework, responsible for orchestrating service deployment across a distributed edge topology. The repository contains source code, Dockerfile definitions, and Kubernetes manifests necessary for containerized and local deployments.

SNIPPET 1: SMO REPOSITORY DIRECTORY STRUCTURE

```

SMO/
├── components/
│   └── service-orchestrator/
│       ├── src/
│       │   └── app.py           # Main FastAPI application for orchestrating services
│       ├── Dockerfile          # Container image for the orchestrator
│       └── requirements.txt     # Python dependencies
├── deployment/
│   ├── deploy/                # Kubernetes YAMLs for deploying orchestrator components
│   └── compose/               # Docker Compose files for local deployment
├── Makefile                   # Automation targets for building and deploying
└── README.md

```

We introduced simple and consistent automation mechanisms for both Kubernetes and local deployments:

To build all components into container images, and optionally push them to a container registry, issue the command according to Snippet 2.

SNIPPET 2: CONTAINER IMAGE BUILD

```
make images
```

To generate and apply the required Kubernetes manifests for deploying the orchestrator into an existing cluster, use the commands according to Snippet 3.

SNIPPET 3: DEPLOY THE SMO INTO AN EXISTING K8S CLUSTER

```
make deploy
kubectl apply -f deployment/deploy/
```

Additionally, to enable rapid testing and debugging we provide the option to spin up all components locally using Docker Compose (Snippet 4).

SNIPPET 4: LOCAL DEPLOYMENT FOR DEBUGGING

```
make local
docker compose -f deployment/compose/docker-compose.yaml up -d
```

Before deploying, users must configure GitHub access credentials by editing:

- `servicecatalog-creds.yaml` (for Kubernetes)
- `docker-compose.yaml` (for local deployment)

2.1.5 Cloud-native deployment targets

As part of our contribution to DESIRE6G, we propose a cloud-native architecture that enables lightweight, Cortex-M-class devices, such as selected switches, edge devices, sensors or custom

compute boards, to participate in the broader programmable infrastructure alongside more capable edge and cloud resources (Figure 8). These devices, despite their limited resources, are treated as managed and trusted endpoints, thanks to a secure onboarding pipeline, cloud-native job orchestration, and lightweight execution mechanisms. This enables new deployment models across energy-efficient edge scenarios, such as localized sensing, real-time control, and ephemeral compute offloading.

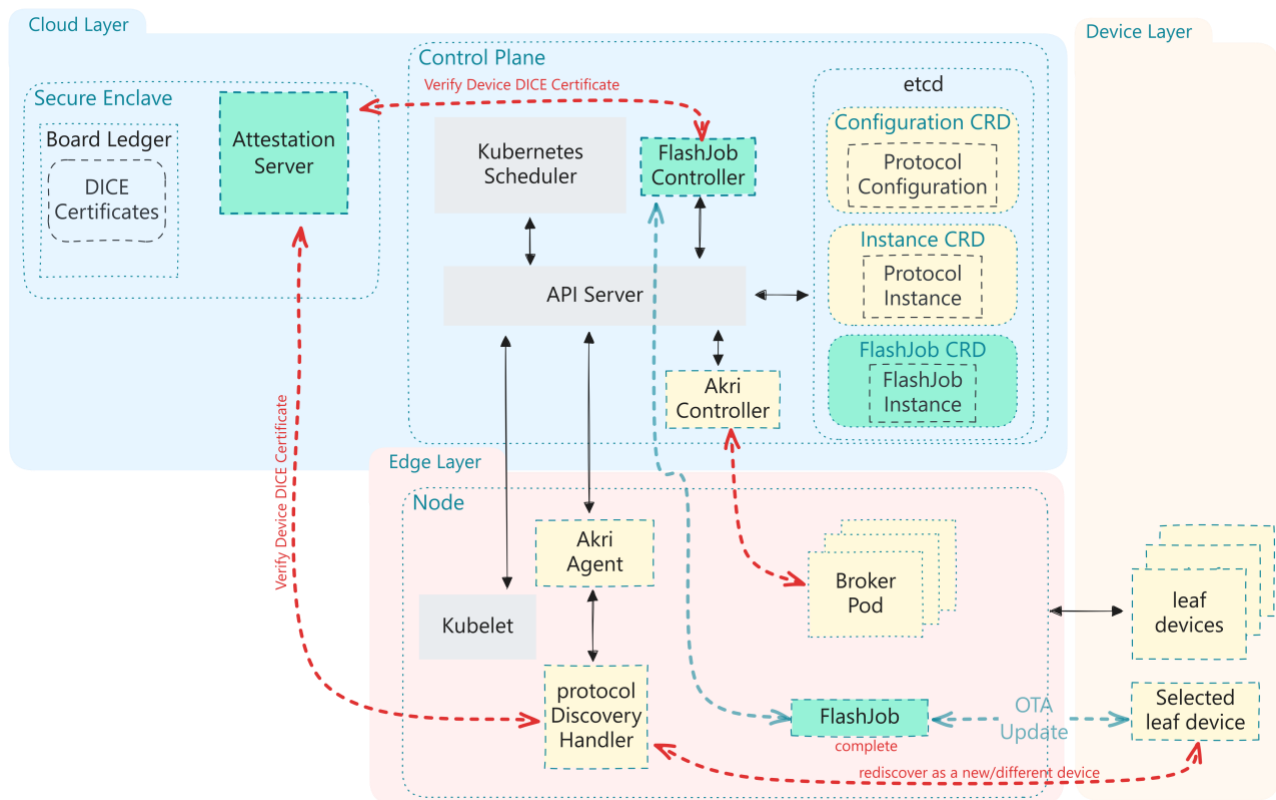


FIGURE 8. ARCHITECTURE FOR ONBOARDING RESOURCE CONSTRAINED DEVICES

The architecture consists of three layers: the cloud/control layer, the edge node layer, and the device layer. Devices initiate onboarding by presenting DICE-based certificates, which are verified against a secure enclave ledger via an Attestation Server. Upon successful attestation, the device becomes visible to the control plane, where Kubernetes-native controllers, such as Akri¹ and our custom FlashJob controller, take over the orchestration process.

¹ <https://github.com/project-akri/akri/>

Within the edge node layer, the Akri Agent continuously runs a protocol-specific Discovery Handler that detects attested devices and registers them as Kubernetes resources. The Akri Controller reconciles these resources into Instance CRDs, each representing a protocol-compliant device or group of devices. In parallel, the FlashJob Controller reacts to the appearance of trusted devices and schedules short-lived workloads (FlashJobs), which are executed on demand through dedicated Broker Pods. This pattern is ideal for workloads that cannot run persistently on constrained devices, such as periodic inference, environmental sampling, or firmware updates & diagnostics.

FlashJobs interact directly with the selected target device and can trigger a secure OTA update when necessary. Updated devices are re-discovered and re-attested, ensuring that their new state is accurately reflected in the system. All metadata (e.g., capabilities, firmware version, trust level) is managed declaratively via Kubernetes CRDs, allowing for fine-grained control by higher-level orchestrators.

From the perspective of the D6G SMO, this architecture exposes a secure, scalable, and extensible interface for interacting with even the most constrained parts of the network. The SMO can query device capabilities, apply policy-driven job scheduling, initiate attestation verification, and trigger lifecycle events (such as updates or decommissions) using standard Kubernetes APIs, through IML. This enables composable service chaining and fine-grained control over distributed edge intelligence, all while preserving the security guarantees and scalability required in the DESIRE6G environment.

2.2 Optimization Engine

This section introduces the Optimization Engine (OE), a key module of the SMO framework designed for generic optimization tasks. During service deployment and service assurance (see D2.2 [15]), the OE interacts directly with the SO, receiving service requests in the form of service graphs annotated with end-to-end SLAs. Leveraging machine learning and artificial intelligence techniques, the OE performs service graph partitioning, providing the SO with partial Service Graphs annotated with their respective partial SLOs. This section presents our initial development blocks of OE. In this case, the OE examines the received unoptimized service and chooses an algorithm from a pool that is best suited in partitioning the service according to the annotated SLAs and SLOs.

2.2.1 AI/ML Workflow

The workflow of the module, seen in Figure 9, can be summarized with 5 steps:

1. Receiving a service graph from the SO, previously annotated by the IBN.
2. Selecting the best-suited partitioning approach from a pool, based on the annotated information (see Section 4.1.3).
3. Execute optimization with the selected algorithm (see Section 4.1.2),
4. Deliver the partitioned service graph to the SO for deployment (partial Service Graphs annotated with their respective SLOs and mappings to DESIRE6G sites).
5. Evaluate periodically the models' performance based on feedback from the other SMO modules.

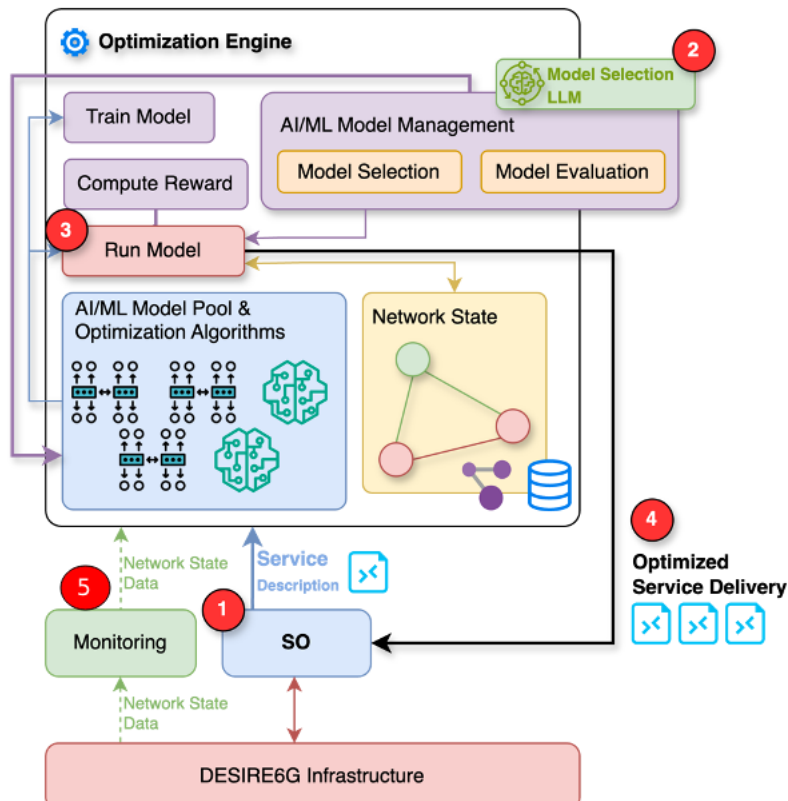


FIGURE 9: OPTIMIZATION ENGINE AI/ML ARCHITECTURE.

2.2.2 Model Pool

The AI/ML Model Pool is a collection of AI algorithms that can be used for service graph optimization. They operate based on distinct Key Performance Indicators (KPIs). In the case of the main use case scenario, a graph partitioning model can be used to minimize E2E latency. Each algorithm targets a specific aspect of performance, enabling that way an efficient and optimized service delivery based on the network conditions, performance evaluation, SLAs and SLOs.

2.2.3 Model Management and Selection

The AI/ML Model Management Component is a collection of tools that analyze the received service descriptions and perform the algorithm selection, usually based on local Large Language Models (LLMs). Additionally, it is responsible for the model performance evaluation after the actuation, giving feedback to the internal mechanism to fine-tune the selection process.

2.2.4 Optimization Subsystem Implementation

The OE is designed with modularity in mind. It consists of various submodules built using Python, with flexibility, maintainability and expandability in mind. An API enables seamless communication with the other SMO modules. Furthermore, the OE features a graphical user interface (GUI) Web Application that allows the administrator to fine-tune and supervise the optimization process in an easy-to-understand way, seen in Figure 10.

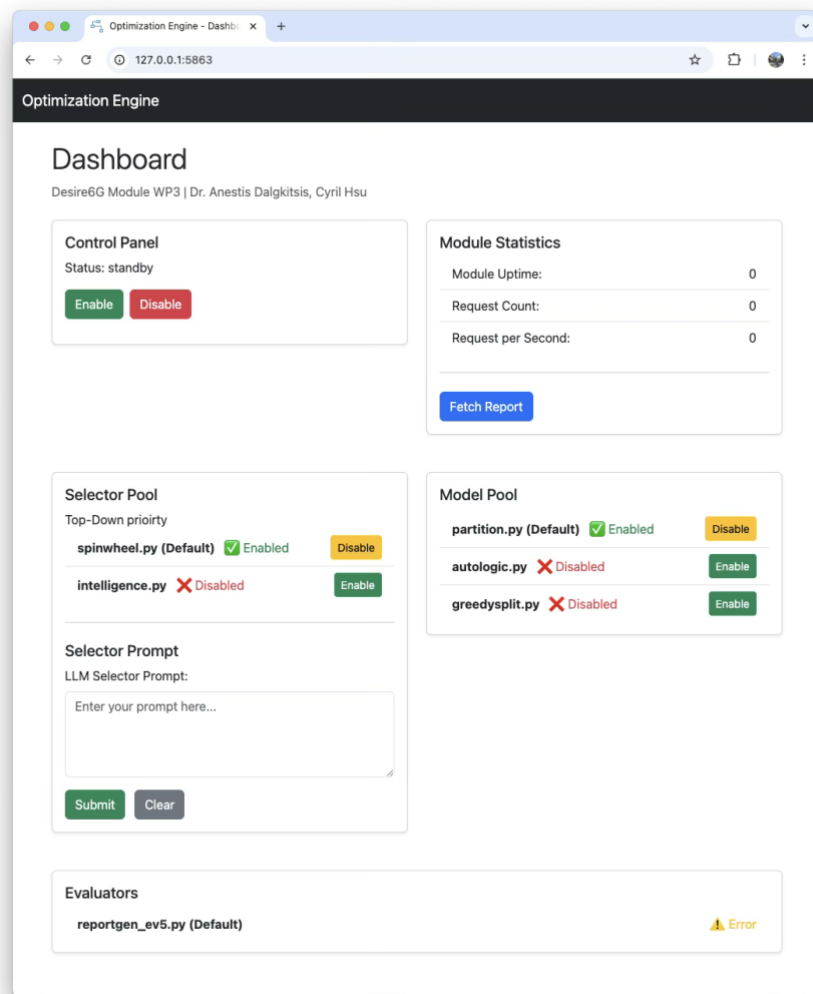


FIGURE 10. OPTIMIZATION ENGINE WEB GUI.

The OE supports plug-and-play optimization models and algorithms with Python, allowing quick experimentation and selection of the most suitable ones for each specific use case.

2.2.5 Encoding/Decoding and Processing Service Requests

The OE receives service graphs from the SO in a predefined and standardized YAML format. Upon receiving the YAML request, decodes the graph into an internal representation using NetworkX [19], that allows the optimization subprocess to execute any algorithm from the pool in a plug-and-play fashion.

```

1  lnsd:
2    ns-instance-id: "11223344-e2a8-4338-bc8c-be685548bad2"
3    ns:
4      name: "Digital Twin Demo"
5      id: "418420e3-e2a8-4338-bc8c-be685548bad2"
6      vendor: "D6G"
7      descriptor-version: "1.0"
8      network-functions:
9        - nf-instance-id: "flowcl-i01"
10          nf-id: "flowcl"
11          nf-name: "Flow Classifier"
12          nf-version: "v1.0"
13          nf-mgmt-network: "shared-mgmt-net"
14        ...
15      application-functions:
16        - af-instance-id: "dt-ros-master"
17          af-id: "ros-master"
18          af-name: "ROS Master"
19          af-version: "1.0"
20          af-mgmt-network: "shared-mgmt-net"
21        ...
22      mas-agent:
23        instance-id: "mas0012"
24        id: "smas001"
25        name: "service_mas"
26        version: "v1.0"
27        ...
28      forwarding_graphs:
29        - member-graph-index: 1
30          graph-name: "uplink"
31          links:
32            - id: "input-1"
33              connection-points:
34                - member-connection-point-index: 1
35                  member-if-id-ref: "sc1-sc2:input"
36                - member-connection-point-index: 2
37                  member-if-id-ref: "flowcl-i01:port-input"
38              ...
39          e2e_delay_budget: "5ms"
40

```

FIGURE 11. SERVICE REQUEST.

After the partitioning phase, the OE encodes the partial service graphs into a list of YAML graphs, to ensure their compatibility with the rest of the SMO modules. Finally, the OE verifies the structure of the

generated YAML files before sending the partial service graphs to the SO to continue with the service deployment/assurance process.

SNIPPET 5: PARTITIONED PARTIAL SERVICE REQUEST YAML.

```

1  s0e:
2    lnsd:
3      ns-instance-id: "11223344-e2a8-4338-bc8c-be685592bad0"
4      ns:
5        name: "Digital Twin Demo Partition 1"
6        id: "418420e3-e2a8-4338-bc8c-be685592bad0"
7        vendor: "D6G"
8        descriptor-version: "1.0"
9        site-id: "d6g-001"
10       network-functions:
11         - nf-instance-id: "flowcl-i01"
12           nf-id: "flowcl"
13           nf-name: "Flow Classifier"
14           nf-version: "v1.0"
15           nf-mgmt-network: "shared-mgmt-net"
16           ...
17         e2e_delay_budget: "2ms"
18
19  s1e:
20    lnsd:
21      ns-instance-id: "11223344-e2a8-4338-bc8c-be685548bad1"
22      ns:
23        name: "Digital Twin Demo Partition 2"
24        id: "418420e3-e2a8-4338-bc8c-be685548bad1"
25        vendor: "D6G"
26        descriptor-version: "1.0"
27        site-id: "d6g-002"
28        network-functions:
29         - nf-instance-id: "nat-i01"
30           nf-id: "nat"
31           nf-name: "NAT"
32           nf-version: "v1.0"
33           nf-mgmt-network: "shared-mgmt-net"
34           ...
35         e2e_delay_budget: "3ms"
36  ...

```

In case of an error in the structure of incoming service request graph or lack of resources in the underlying Desire 6G sites that prevent the optimization procedure, the OE responds with a dedicated error message to the SO module.

1 Failed: "Not have enough resources to host service."

FIGURE 12. ERROR MESSAGE DURING PARTITIONING.

2.3 IBN Management

Within the scope of intent-based network management in the project, the focus is on handling new service requests from customers and translating these requests to intent RDF form. The IBN activates the new service intent by working with the OE for service graph and decomposition based on existing

slices. The continuation of the work mentioned in D3.2, along with additional research and innovations, is described in the following sections.

2.3.1 SLA-to-Intent Translation

The SLA-to-Intent Translation Framework is a component that takes service requests from the customer in natural free-text form and translates them into an intent format that the network can understand in DESIRE6G architecture. It acts as a bridge between the customer and the network, automating mutual understanding and enabling a key step toward zero-touch networking.

As illustrated in Figure 13, the SLA-to-Intent Translation Framework comprises key components: customer demand input, API interface, SLA processing and Service Level Objective (SLO) generation module, vector database, and intent generator.

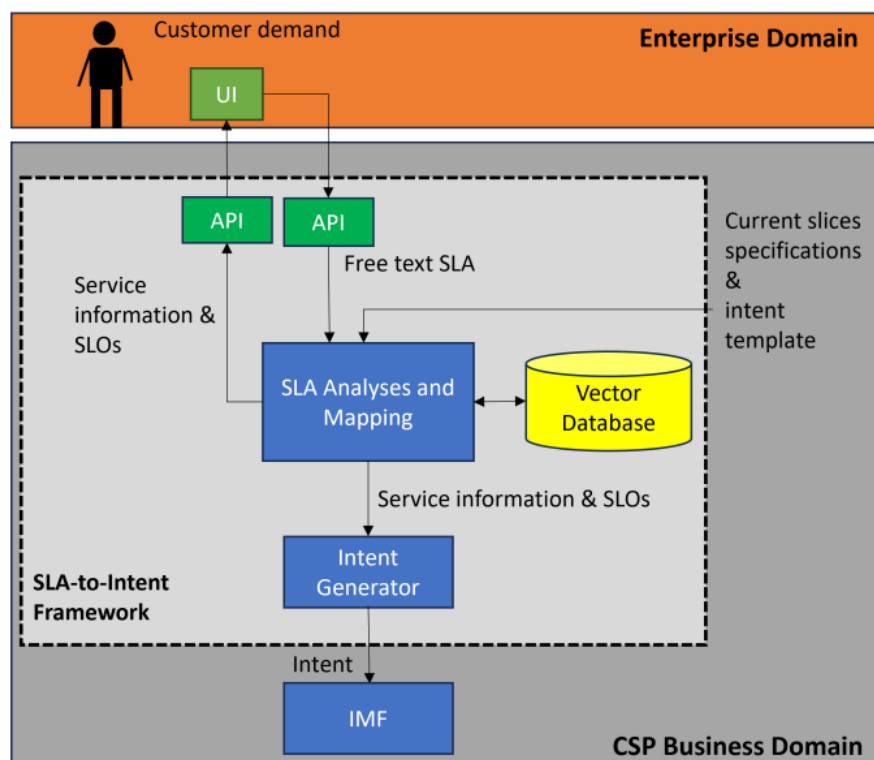


FIGURE 13. SLA-TO-INTENT TRANSLATION FRAMEWORK

The user interface accepts customer demands in natural free text format, enabling easy interaction between the SLA consumer and the framework. Built with Python (Flask) and JavaScript, the application uses RESTful APIs for efficient backend communication. This design ensures smooth data exchange, high performance, and scalability for future integrations.

SLA analyses and mapping plays a key role by extracting important metrics from customer demands and SLAs. It uses vector embeddings to characterize existing CSP services and align them with customer needs. Large Language Models (LLMs), such as OpenAI GPT models [20], analyze free-text inputs to identify essential network-related keywords and convert them into network-specific SLOs.

GPT's transformer architecture enables it to generate text embeddings numerical vectors representing the meaning of text which are used to measure similarity between SLA requirements and predefined network slices. The Retrieval-Augmented Generation (RAG) framework [21] combines GPT with external knowledge sources like vector databases to enhance SLA-to-Intent translation.

Vector databases such as ChromaDB [22] store and manage these embeddings, allowing efficient semantic search and quick retrieval of relevant data. This supports accurate and relevant AI-driven responses.

To standardize SLA handling, fundamental SLA elements including keywords, units, calculations, and constraints—are organized into an extended tabular format based on the ETSI model [23]. This table covers all SLOs before, during, and after service provision, including categories like sales, provisioning, upgrades, support, billing, and service management. It details metrics such as availability, integrity, time, capacity, reliability, flexibility, usability, and security, along with expectations, penalties, and specific conditions. These tables are stored in JSON or YAML formats within a vector database and tagged by use case. New SLA metrics are compared against existing service metrics for similar offerings, enabling effective SLO definition for OSS and BSS layers.

The intent generator takes the finalized SLOs and customer information and automatically creates the intent in RDF format to be sent to the relevant IMF by using one-shot learning. One-shot learning [24] is a machine learning technique where a model learns to perform a task or grasp a concept from just a single example. GPT models demonstrate strong one-shot learning capabilities due to their extensive pretraining on diverse datasets and their ability to generalize patterns from minimal input. In this approach, GPT receives a clear, concise example embedded within a well-structured prompt that sets the task context. Afterward, a new query is presented, prompting GPT to infer the underlying rules and apply the learned behavior to generate an appropriate response. This ability enables GPT to generalize accurately from limited data, making it highly effective for tasks that require minimal training.

Table 2 presents the entire framework in the form of pseudo-code. This high-level representation outlines the proposed algorithm step by step, as detailed below.

- Steps 1-2: the customer inputs their requirements and desired service level expectations in free text via the web interface. This input is sent through an API to the backend for processing and SLOs generation.
- Step 3: the backend receives the input, tokenizes the text, and converts it into vector embeddings.
- Step 4: a chromadb client is created to manage embeddings and data storage.
- Step 5: predefined SLA slice characteristics are fetched from a YAML data source, extracted, and embedded as vectors. These embeddings are stored in a chromadb collection named “slice usecase collection.” If the collection already exists, it is deleted to avoid duplicates.
- Step 6: the system queries the collection to find the best matching slice by comparing the customer’s SLA embeddings with the predefined slices.
- Step 7: the matched slice and corresponding SLOs are sent back to the ui for customer review.
- Step 8: the customer reviews the proposed SLOs and metrics. They can accept them or provide more detailed input for refinement, returning to earlier steps if needed.
- Step 9: once the customer confirms the SLOs, the backend receives confirmation and sends the finalized slice definition and SLOs to the intent generator.
- Steps 10-12: the intent generator uses a rdf intent template and applies one-shot learning to generate the structured intent, incorporating service specs, top SLA keywords, and confirmed SLOs.
- Step 13: the generated intent is sent to the BSS IMF for integration with business processes such as billing and service monitoring.

The processes of gathering requirements from the customer, mapping them to existing services, identifying key metrics for service demands, and generating the corresponding intent are automated. Experiments have been conducted with different examples for three distinct slices (eMBB, URLLC, and mIoT), tested across many different use cases and the results demonstrated successful matching, accurate keyword extraction, and effective translation into the intent language.

TABLE 2. SLA-TO-INTENT TRANSLATION ALGORITHM

Input: Free text customer demand, slice definition and metrics templates, keywords, sample intent template
Output: Intent
UI/API
1: Submit use case requirement via <i>UI</i> (customer)
2: Send use case requirements to <i>SLA Analyses & Mapping</i> via an <i>API</i>
SLA Analyses & Mapping
3: Get free text SLA
Tokenize & Embed results in a vector
4: Create a " <i>CromaDB</i> " client
5: Fetch predefined slices
Extract keys & values
Generate embeddings in vector
Define collection name
List all collections in the vector database
Check if available collection (exists/not exist)
If yes: delete it to prevent duplicates
If no: store vectors in " <i>CromaDB</i> "
6: Query " <i>CromaDB</i> " slice collection (find matching slice to user demand)
7: Send matched slice's definition & SLOs to " <i>API/UI</i> "
API/UI
8: Customer checks SLO specification & metrics on SLO tab (demand validation)
If matched, submit SLO to create intent
If unmatched, go to <i>step1</i> (provide more details)
SLA Analyses and Mapping
9: Send slice definition & SLOs to " <i>Intent Generator</i> "
Intent Generator
10: Get rdf sample template applying one shot learning
11: Fetch " <i>top – sla – keywords</i> "
12: Generate intent
13: Sent intent to BSS IMF

2.3.2 Implementation and Workflow of the Intent Management Function (IMF)

The IMF retrieves the necessary base slice information from the catalog and captures customer requirements in intent format. It then provides the SO with service details, non-functional requirements, and service graph information in the form of Network Service Descriptor (NSDs) and intents as illustrated in Figure 14.

IMF is prepared as a microservice developed in Python 3.11.9. The sub-functions are as follows:

- A script is developed to use RabbitMQ for communicating with other SMO functions, such as the SO and Catalog.
- A function was created to get service information from the catalog. It sends a message to a RabbitMQ queue called catalog_queue, asking for a list of all available services (with the action

'list_services'). Then it waits for a reply and returns the list, which includes service JSON files defined in the catalog.

- Another function is created to send the new service request to SO. The function reads a YAML file, which contains the NSD to be sent to the SO. It converts the content to a string, connects to RabbitMQ, and sends it to the orchestrator_queue for the SO to process. Once the SO receives the NSD or intent, it forwards the request to the OE to decompose the service graph based on the service requirements.
- A fast API and sub-function are designed for communication between IBN and SLA-to-Intent translator.
- The main function is defined to match incoming service requests with existing slice metrics. Once a matching slice is identified and accepted as the base slice, the new service requirements and related information are incorporated. The base slice information, service graph, and customer requirements are used to generate an NSD in YAML format and an intent in RDF format, which are then sent to the SO.

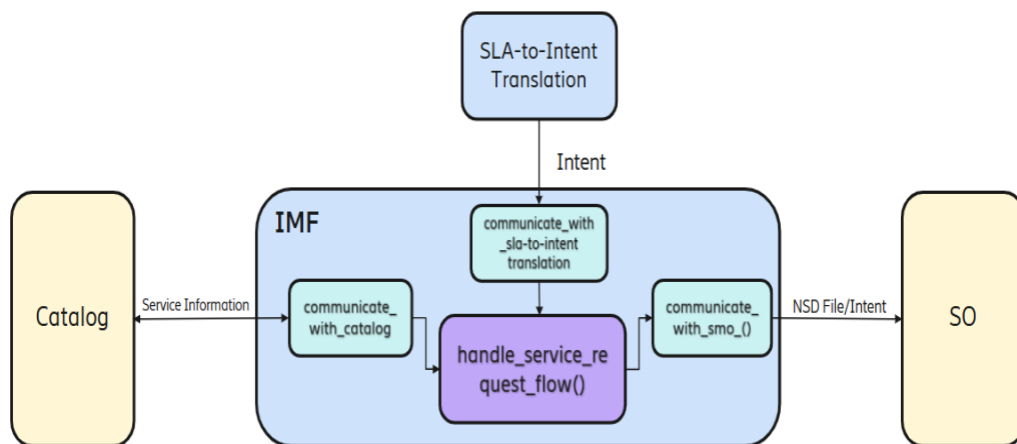


FIGURE 14. IMF MICROSERVICE WORKFLOW WITHIN SMO ENVIRONMENT

The microservice is then containerized and prepared for deployment on the 5TONIC platform.

The Service Catalog definition of the slices is prepared based on TM Forum standards and includes detailed information about slices. The base service chain of the slices, taken from the Desire6G demo, are also added to the catalog. Each slice includes the name, service type, service features, related customer information, usage area, and service chain details. NSD files according to slice catalog information and customer information and its requirements as a Yet Another Markup Language (YAML) format are prepared.

In summary, the customer request is translated into intent format, the corresponding slice from the catalog is mapped, and the information is forwarded to the SO. The data shared with SO includes service requirements, the service graph, and service details. As a result, SO receives the necessary information for resource management and decomposition across subdomains and proceeds to activate the service. For future work, it is proposed that catalog queries also become intent-specific, allowing the service initiation process to be fully aligned with intent-based management principles. Additionally, implementing intent-based management within subdomains can further enhance automation, flexibility, and end-to-end orchestration across the entire system.

2.3.3 Intent Extension Models for Cross-Domain Automation in Digitalized

Manufacturing

The digitalization of factories demands flexible, autonomous, and highly connected environments, with private networks playing a crucial role. This approach enables automation by translating high-level business goals into actionable network behavior, minimizing manual control.

The enterprise domain and its digitalization are focused on better understanding the associated requirements. Initially, the types of network requests that could be received from customers and their possible representation in an intent format were analyzed. Subsequently, the potential application of intent-based management within enterprise environments, as illustrated in Figure 15, was explored. For this purpose, the definition of intents was examined. This analysis particularly considered scenarios where the network might need to respond to or be impacted by enterprise-side events, such as production increases.

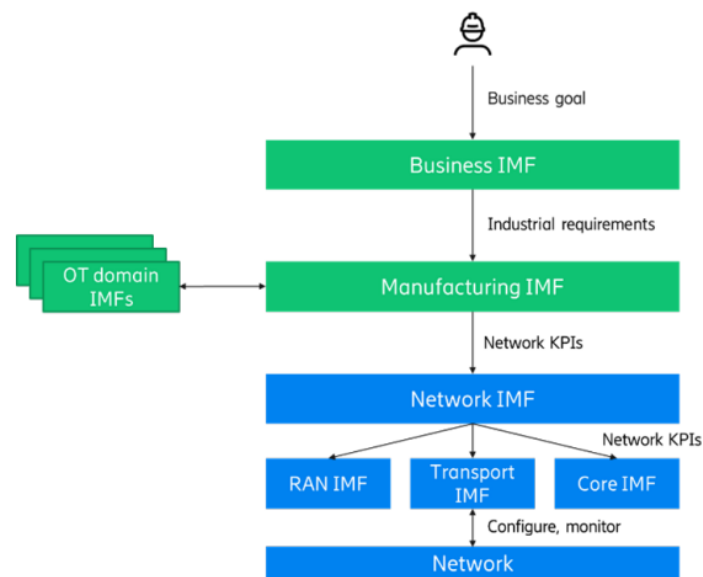


FIGURE 15. INTENT-ORIENTED MANAGEMENT HIERARCHY IN INDUSTRIAL ENVIRONMENTS

The scope of intent handling wireless industrial automation is shaped by both functional and non-functional requirements. For smart manufacturing use cases, such as collaborative robots, non-functional needs must be expressed through precise KPI clusters. ISO 22400 defines widely accepted industrial KPIs (e.g., mean time to failure, availability) [25], while 5G-ACIA outlines key metrics for automation, including cycle time, availability, and security [26]. These KPIs are essential for effective intent-based automation and performance evaluation.

Intent extension models are proposed to illustrate how the TM Forum’s common intent model can be specialized for industrial domains and seamlessly integrated into the intent model federation. Using RDF representation, these extensions define classes, properties, expectations, and metrics tailored to industrial processes such as equipment maintenance and performance optimization. A key concept is the intent manager scope, which outlines the responsibilities and operational range of the manager. While core capabilities must be standardized, optional domain-specific additions can be collaboratively defined by Standards Developing Organizations, making strict governance unnecessary.

To illustrate, a use case for predictive maintenance in Industry 4.0 involves tracking device location and downtime. For example: “For 80% of devices, downtime must not exceed 20 seconds, operations must occur in specific regions, and power consumption must remain minimal.” This intent is formally represented in RDF, referencing KPI-based parameters such as *kpi:downtime*, *kpi:powerConsumption*,

and *kpi:deviceLocation*. These parameters are linked through properties such as *icm:hasParameter*, as demonstrated in Listing 1. Each element is defined using dedicated namespaces, enabling integration with existing KPI catalogs through standard URIs.

```
@prefix icm: <http://tio.models.tmforum.org/tio/v2.0.0/IntentCommonModel/> .
@prefix kpi: <http://www.sdo3.org/KPI/Version2/> .
@prefix ind: <http://www.sdo3.org/industry/Version2/> .
@prefix : <http://www.operator.org/IntentNamespace/intent20220322_12345/> .
----- Parameters -----
:ProductionLine
  a rdfs:Class ;
  rdfs:subClassOf :IndustrialFunction ;
  rdfs:label " ProductionLine class" .
# Defined properties kpi:powerConsumption ,
#                   kpi:downtime ,
#                   kpi:deviceLocation ,
#                   kpi:accidentRate ,
#                   kpi:efficiencyIndex ,
#                   kpi:deviceAvailability .
kpi:powerConsumption
  a rdf:Property ;
  rdfs:label "Power " ;
  rdfs:comment "power consumption value of the production line".
kpi:downtime
  a rdf:Property ;
  rdfs:label "Down Time " ;
  icm:unitOfMeasurement :second ;
  rdfs:comment "The duration of the downtime" .
kpi:deviceLocation
  a rdf:Property ;
  rdfs:label "Location" ;
  rdfs:comment "current device location" .
kpi:efficiencyIndex
  a rdf:Property ;
  rdfs:label "Efficiency Index " ;
  rdfs:comment " Overall equipment efficiency index " .
kpi:deviceAvailability
  a rdf:Property ;
  rdfs:label "Availability" ;
  icm:unitOfMeasurement :Percent ;
  rdfs:comment "Current device availability" .
```

LISTING 1. INTENT EXTENSION MODEL FOR INDUSTRY

Expectations are defined for device availability, downtime duration, device location, and power consumption, demonstrating how metrics from diverse sources can be combined flexibly. In the industrial intent, Production Target 1 defines three conditions. The first condition limits the allowable downtime duration. The second restricts devices in Device Group 1 to operate only within Region A. The third specifies the power consumption requirements, as illustrated in Listing 2.

```

@prefix icm: <http://tio.models.tmforum.org/tio/v2.0.0/IntentCommonModel/> .
@prefix kpi: <http://www.sdo3.org/KPI/Version2/> .
@prefix ind: <http://www.sdo3.org/industry/Version2/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix log: <http://tio.models.tmforum.org/tio/v3.2.0/LogicalOperators/> .
@prefix : <http://www.operator.org/IntentNamespace/intent20220322_12345/> .
# --- Intent -----
:IndustrialIntentExample
a icm:Intent ;
rdfs:comment " 80\% of the devices shall have downtime < 20s,
shall be in Region A, and shall operate with minimum power consumption.
log:allOf ( :industrial-intent )
.
# --- Targets -----
:ProductionTarget1 a icm:Target ;
rdfs:member :Devicegroup1 , :Devicegroup2 ,
:Devicegroup3 , :Devicegroup4 .
# --- Expectations -----
:industrial-intent
a :ProductionExpectation ;
icm:target :ProductionTarget1 ;
:percent 0.8 ;
log:allOf ( :condition-1 :condition-2 :condition-3 ) .
:condition-1
a icm:Condition ;
log:allOf ( :quan:smaller ( [ met:lastValue
( kpi:downtime ) "20s"^^quan:quantity ] ) ) .
:condition-2
a icm:Condition ;
set:elementOf ( ind:Devicegroup1 ind:RegionA ) .
:condition-3
a icm:Condition ;
quan:smaller ( [ met:lastValue ( kpi:powerConsumption ) 0 ] ) .

```

LISTING 2. CUSTOMER INTENT

Beyond the basic use case, intent extension models can include specialized industrial expectations like: *ProductionExpectation* as shown in Listing 3. This can be further extended to: *DeviationExpectation*—triggering data collection or corrective actions when error rates exceed thresholds—and *EfficiencyExpectation*, which sets constraints to optimize production efficiency by maximizing output and minimizing production time.

```

:ProductionExpectation
a rdfs:Class ;
rdfs:subClassOf icm:Expectation ;
rdfs:comment "Abstract class of expectation that have instance target".
:DeviationExpectation
a rdfs:Class ;
rdfs:subClassOf :ProductionExpectation ;
rdfs:comment "Expectation to allow deviation from standard procedures".
:EfficiencyExpectation
a rdfs:Class ;
rdfs:subClassOf :ProductionExpectation ;
rdfs:comment "Expectation to provide a certain level of
production efficiency".

```

LISTING 3. INTENT EXTENSION MODEL FOR INDUSTRIAL EXPECTATIONS

Key requirements of emerging industrial use cases are identified, and an extension to the TM Forum's common intent model is proposed to enhance support for intent-driven network management in industrial environments. It introduces a tailored vocabulary and hierarchical structure for handling intents, addressing the current gap in semantics for industrial applications. The main contribution is the development of an intent extension model that enables end-to-end autonomous operations by bridging industrial requirements and network capabilities through intent-based transitions. Future directions include expanding the model's vocabulary to cover additional use cases and refining intent translation to enhance the performance of the automation framework [27].

2.3.4 Autonomous Decision-Making Framework for Supporting Multiple Services in the IMF

Each service type has specific KPIs that must be satisfied simultaneously for effective operation, and these KPI requirements are managed at the IMF level. The challenge lies in efficiently allocating limited network resources to meet these diverse and sometimes conflicting demands. As the number and complexity of service requirements grow, automating decision-making at the IMF becomes increasingly essential.

The IMF process involves a hierarchy of specialized agents, including proposal, prediction, and evaluation agents, that collaboratively enable adaptive network optimization (Figure 16). These agents generate candidate actions, predict their impact using techniques such as Gaussian Process regression, and assess utility under uncertainty via Bayesian Optimization. This architecture supports informed, autonomous decision-making in dynamic and stochastic network environments.

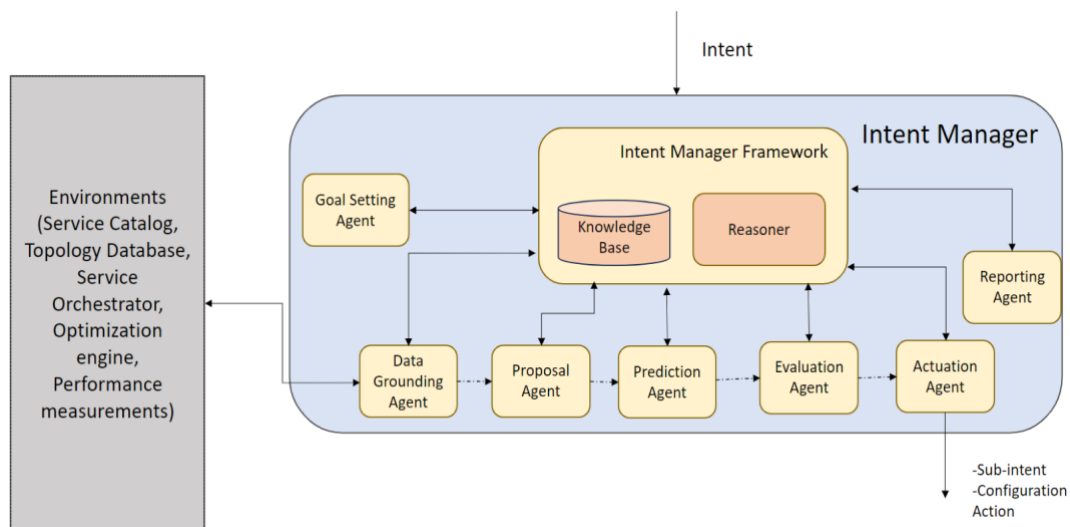


FIGURE 16. INTENT MANAGEMENT FUNCTIONS

A method is being developed for autonomous decision-making in communication networks, based on collaboration between two specialized agents. The Proposal Agent generates multiple network action proposals, each containing network modifications, predicted performance, and uncertainty metrics. The Evaluation Agent then assesses these proposals using utility metrics in a structured two-step process.

In the first step, proposals that satisfy both the uncertainty and utility thresholds are prioritized according to their predicted performance. If none meet these thresholds, proposals from agents with the highest average uncertainty are selected instead. These are not retrained but rather chosen to provide dynamic metrics that feed into the optimization engine's self-training, thereby supporting continuous learning.

This dual mechanism ensures a balance between exploiting reliable actions and exploring uncertain ones, which improves predictive accuracy and enables optimized, uncertainty-aware network decisions. The detailed workflow describing this interaction is outlined in Steps 1–13 below.

Performed by Proposal Agent:

1. Obtain network performance data reflecting the current network state.
2. Determine a surrogate function based on the performance data that models:
 - The predicted relationship between available actions and resulting network performance metrics.
 - The uncertainty associated with each action's predicted performance.
(This may involve fitting a probabilistic model such as Gaussian Process regression.)

3. Select a proposed action by analyzing the surrogate function, including:
 - The predicted value of the network performance metric if the action is taken.
 - The predicted uncertainty metric associated with this action.
4. Create an action proposal containing the proposed action, predicted performance value, and uncertainty value.
5. Send the action proposal to the Evaluation Agent.
6. Optionally, after sending the proposal, update the surrogate function and uncertainty metrics with new performance data as it becomes available.

Performed by Evaluation Agent:

7. Receive multiple action proposals from one or more Proposal Agents.
8. For each action proposal, calculate a utility metric that reflects the predicted network performance across multiple metrics.
9. Check if any proposal meets both:
 - An uncertainty criterion (i.e., the uncertainty metric is within acceptable limits).
 - A utility criterion (i.e., the predicted performance utility meets required thresholds).
10. If one or more proposals satisfy both criteria:
 - Apply a first selection criterion to choose the proposal with the best predicted performance (highest utility).
 - Prefer proposals with lower uncertainty.
11. If no proposal satisfies both criteria:
 - Apply a second selection criterion to choose the proposal from the Proposal Agent with the highest average uncertainty among its available actions (favoring exploration of uncertain areas).
 - Additionally, bias selection toward proposals with potential for the highest achievable utility improvement.
12. Select the final action proposal according to these criteria.
13. Send the finalized action to the Actuation Agent to trigger the implementation of the proposed network property modifications.

This agent-based approach enables continuous learning and adaptation, allowing dynamic optimization of network configurations based on performance goals and prediction confidence. By combining

predictive modeling with decision-making under uncertainty, the collaborative framework supports intelligent network management. The Proposal Agent continuously analyzes network data to generate informed actions, while the Evaluation Agent holistically assesses them to balance performance and confidence. Dual selection criteria ensure both exploitation of known optimal actions and exploration of uncertain ones to enhance future decisions. Aligned with intent-based networking, this architecture automates decision-making to meet service objectives and adapt to changing conditions.

In essence, the aim is to develop intelligent, self-managing networks that autonomously predict, evaluate, and implement optimal configurations to efficiently and reliably meet diverse service KPIs.

2.4 DLT Federation

This section provides the final implementation and functional details of the DLT-based Federation, a key module of the SMO framework designed to support service federation between multiple administrative domains (i.e., service providers with the SMO framework and DESIRE6G infrastructure).

2.4.1 DLT-based Federation Implementation

The DLT-based Federation module employs Distributed Ledger Technologies (DLTs), in particular blockchain and smart contracts, to facilitate secure, private, fast, and dynamic interactions between administrative domains in the federation process. This functionality is crucial for ensuring service continuity in dynamic scenarios (e.g., unexpected surges in service demand, urgent scaling requirements, or infrastructure failures), as it allows service providers to orchestrate services from other domains on-the-fly without prior agreements, thereby rapidly extending their local capabilities to meet the specific needs of vertical customers. By employing blockchain, we overcome the limitations of legacy federation solutions [28], which typically rely on pre-established SLAs between service providers (suitable only for static environments), require manual operations that introduce delays and complexity, incur additional overhead costs, and in some cases depend on middle-men or third-party intermediaries, thereby raising concerns related to trust, privacy, and security. Further details on the existing research gap in dynamic service federation and how blockchain addresses these technical challenges are provided in Deliverables D3.1 and D3.2.

The integration of the DLT-based Federation module into the DESIRE6G architecture is illustrated in Figure 17. We adopt a permissioned blockchain network, where each administrative domain can

participate by deploying a DLT-based Federation component connected to the East/Westbound Interface (E/WBI) of the SO. This design addresses the well-known limitations of public, permissionless blockchains, such as slow transaction processing times (i.e., validation and inclusion in a block), high operational costs (i.e., fees for executing smart contract operations), and the lack of control over network membership, which makes it difficult to exclude potentially malicious participants from the federation [29]. In contrast, permissioned blockchains are designed for enterprise environments, offering faster transaction times due to fewer participants and more efficient consensus algorithms, lower operational costs, and built-in mechanisms for identity and access control. In the context of mobile network infrastructures like DESIRE6G, permissioned chains are more appropriate to enable fast interactions across federated domains, while ensuring secure and controlled operation.

The DLT-based Federation is composed of two main containerized submodules: the **Blockchain Manager** and the **Blockchain Node**. The Blockchain Manager acts as a bridge between the SO and the blockchain layer, translating federation-related operations – such as domain registration, service announcements, bid submissions, or deployment information exchanges – into blockchain transactions, and subscribes to smart contract events to notify the SO of relevant updates. The Blockchain Node connects the domain to the blockchain network, participates in the consensus process, and maintains a synchronized local copy of the distributed ledger.

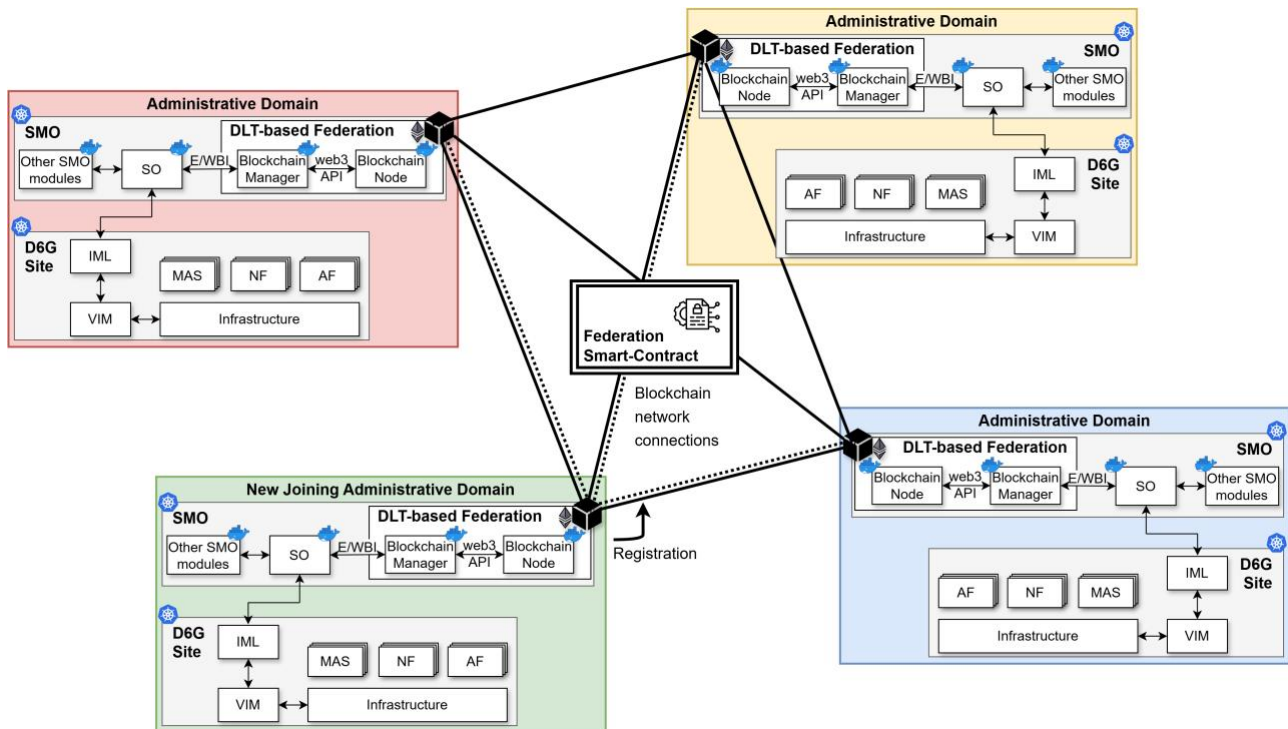


FIGURE 17. INTEGRATION OF DLT-BASED FEDERATION INTO THE DESIRE6G ARCHITECTURE

A generic **Federation Smart Contract (SC)** is deployed on the blockchain to serve as a distributed authority that manages all federation procedures. These smart contracts are autonomous, programmable applications that enforce business logic and rules, similarly to traditional contract agreements. The design of the Federation SC is fully transparent and crucial for safeguarding the privacy of sensitive information while overseeing federation procedures for all participating domains.

Each domain running a blockchain node execute the same instance of the Federation SC simultaneously and interacts with it by submitting cryptographically signed transactions, which are immutably stored on the distributed ledger.

The implementation of the DLT-based Federation module relies on the following key technologies:

- **Blockchain Manager:** Developed using the **FastAPI** web framework, it provides RESTful APIs that expose federation operations to the SO and uses the **web3.py**² library to connect to the blockchain node through the Ethereum JSON-RPC API³ over HTTP(S) and/or WebSocket, supporting real-time subscription to smart contract events, state queries, and transaction submission.
- **Blockchain Node:** Runs a private, permissioned instance of **Go-Ethereum (Geth)**⁴, the official Ethereum client, to connect the domain to the blockchain network. Ethereum was selected as the blockchain platform due to its maturity, extensive documentation, active development community, and flexibility to migrate to a public Ethereum network in future stages if broader interoperability is required.
- **Federation SC:** Written in **Solidity**⁵ programming language, the smart contract is compiled and deployed on the blockchain using the **Truffle**⁶ development framework.

² <https://web3py.readthedocs.io/en/stable/>

³ <https://ethereum.org/en/developers/docs/apis/json-rpc/>

⁴ <https://geth.ethereum.org/docs>

⁵ <https://docs.soliditylang.org/en/v0.8.30/>

⁶ <https://archive.trufflesuite.com/docs/truffle/>

2.4.2 DLT-based Federation Operation

Figure illustrates the operational workflow of the DLT-based Federation module in a representative scenario where an Application Function (AF) experiences a malfunction (e.g., the allocating DESIRE6G site can no longer fulfill its operational criteria). In the absence of sufficient local resources, federation is triggered via the blockchain to migrate the AF to an external provider domain capable of meeting the service requirements.

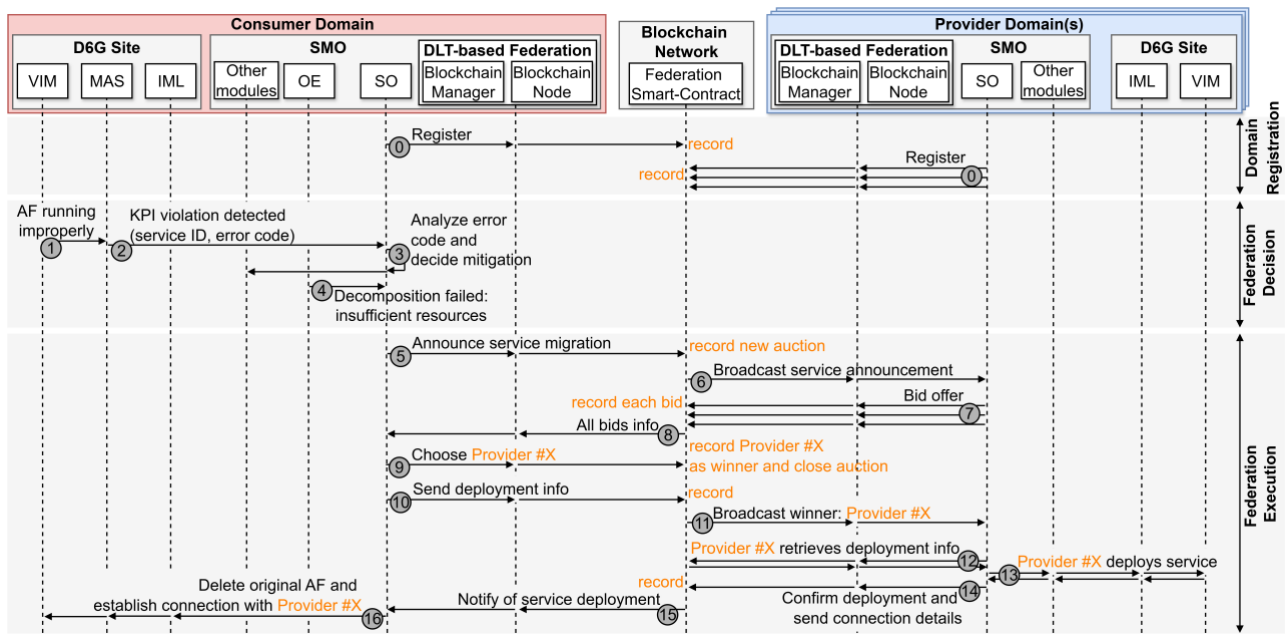


FIGURE 18. OPERATIONAL WORKFLOW OF DLT-BASED FEDERATION (EXAMPLE: SERVICE MIGRATION ACROSS MULTIPLE DOMAINS)

All administrative domains participating in the federation are pre-registered in the Federation SC deployed on the blockchain (step 0), and each is uniquely identified by its blockchain address. Once registered they are ready to consume or provide federated services.

When the AF starts to malfunction, an alert is sent to the SO from the MAS framework layer (steps 1 and 2). This alert contains a service ID and an error code (e.g., "insufficient resources", "site down"), indicating that the MAS is unable to resolve the issue autonomously. The SO first attempts to redeploy the service to another site within the same domain, following the normal service deployment process described in D3.2 (step 3). However, when the OE runs its optimization algorithms to find the proper DESIREG site(s) for re-deployment, it fails to find available resources and returns an error that triggers the federation process on the blockchain (step 4).

The blockchain-based federation process begins when the SO, acting as a consumer domain, submits a service announcement transaction to the Federation SC through the Blockchain Manager (step 5). This announcement specifies the migration requirement along with the necessary resources and QoS constraints. Potential provider domains subscribed to federation events receive this service announcement (step 6) and respond by submitting transactions containing their bid-offers (service prices), following a reverse-auction model (step 7). The consumer domain evaluates the received bids and selects the most suitable provider based on the internal policies (e.g., lowest price, first offer received, etc) (steps 8 and 9). It then shares the necessary deployment information (e.g., endpoints for retrieving the application descriptor and establishing data plane connectivity) with the selected provider (step 10). The provider domain is notified of the selection, retrieve the deployment information from the Federation SC, and proceeds with deploying the federated service (steps 11, 12 and 13). Upon successful deployment, the provider submits a confirmation transaction that includes service instantiation details and connection parameters (step 14). Finally, the consumer domain receives this confirmation and establish data plane connectivity with the provider (steps 15 and 16). Note that the original AF remains operational until the new instance is fully deployed and confirmed at the provider domain, after which it is safely deleted.

2.5 Data lake synchronization

This section reports the implementation and functional details of the mechanism that has been designed to synchronize monitoring data from D6G sites and the central cloud. The module envisages the injection of data in the SMO data lake, making the metrics available for further centralized data processing.

2.5.1 Reference scenario

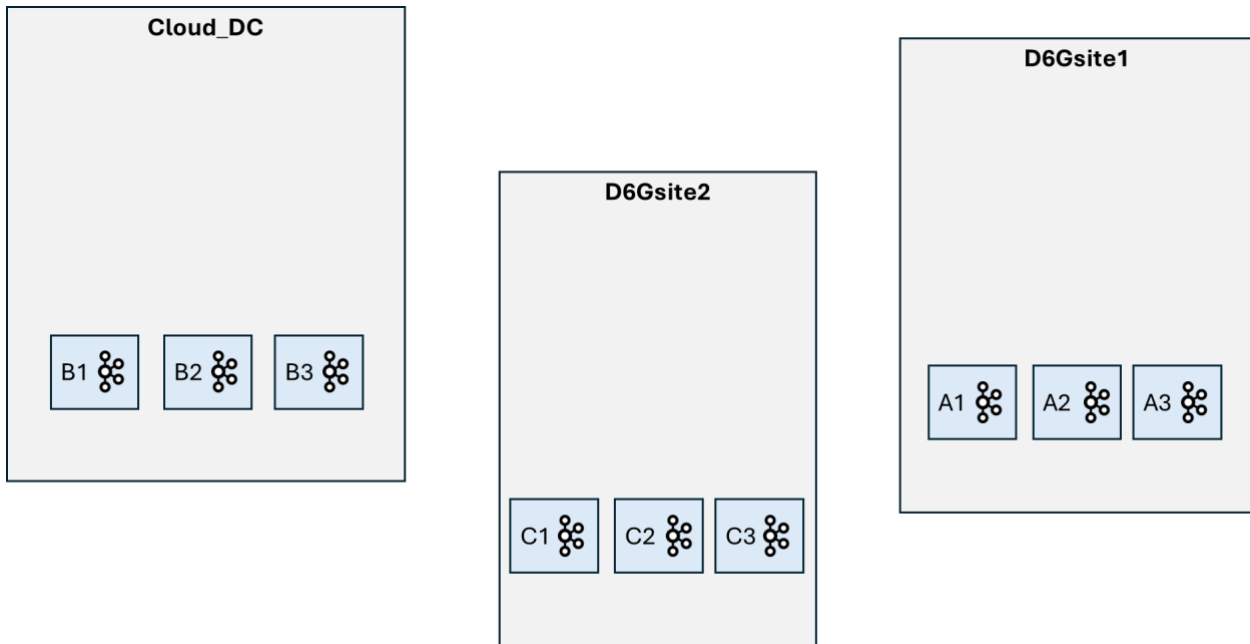


FIGURE 19. REFERENCE SCENARIO

Figure 19 shows the considered reference scenario where the infrastructure is composed by multiple D6G edge sites and a centralized cloud DC. Each site (both edge D6G or central cloud) exploits an Apache Kafka publish subscribe method for the local distribution of monitoring metrics. Each Kafka cluster is composed of multiple brokers (A1, A2, A3 in D6Gsite1, B1, B2, B3 in the central cluster, and C1, C2, C3 in D6Gsite2). In this way, local producers can generate monitoring metrics related to specific services or the infrastructure. By relying on consumers, local routines can consume in real-time the monitoring data to make decisions and/or trigger the service reconfiguration.

2.5.2 Data synchronization

In order to share data among different sites, a specific method has been designed to synchronize the data among Kafka clusters.

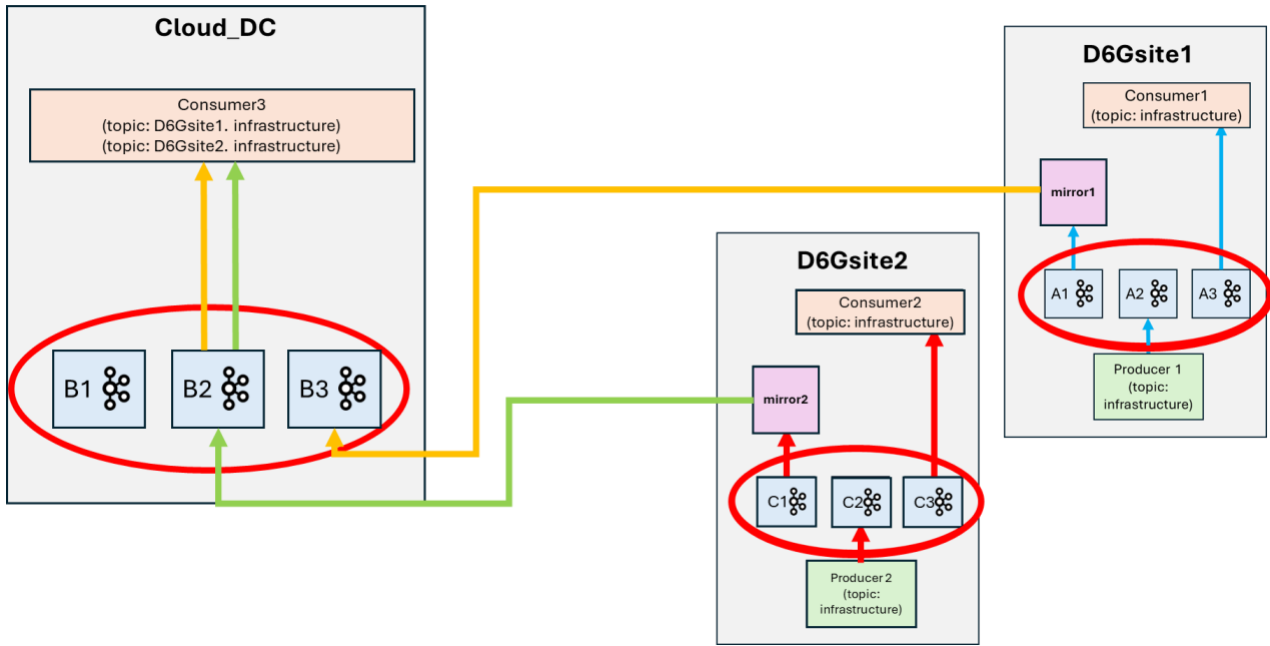


FIGURE 20. DATA SYNCHRONIZATION APPROACH

Figure 20 shows the considered solution for data synchronization. In particular, the figure shows that in each of the two edge D6G sites (D6Gsite1 and D6Gsite2) a producer is enabled to periodically report metrics related to the status of the cloud resources (i.e., the available RAM, CPUs and cores, the number of nodes in the cluster, etc.). This information is pushed into a specific topic highlighted with green and yellow (in the example above “infrastructure” topic). A consumer can locally consume this information, enabling AI/ML driven reconfiguration/routines. In each of the edge sites, a special module has been activated. The approach proposed in [59] has been adapted and revised to cope with D6G requirements. In particular, novel cluster components (called mirror) are enabled to synchronize the data pertaining to a specific topic in the edge site to one in the cloud cluster. The mirror can be configured to act as unidirectional or bidirectional mode and to select specific topics to be redistributed to remote clusters. In our case, the mirror has been configured as unidirectional mode, with data synchronized from the edge clusters to the cloud cluster.

In the figure, an example of consumer in the cloud cluster is shown (i.e., Consumer3). This consumer, attached to the cloud cluster, is able to retrieve data not only from the local topic, but also from the remote topics (i.e., D6Gsite1.infrastructure and D6Gsite2.infrastructure), being able to enable E2E routines encompassing even edge D6G sites. This procedure is particularly relevant for the topology management process, enabling the possibility to highlight at the central SMO the resource availability of all the D6G edge sites, periodically updating the information present in the Topology DB.

3. Service Optimization and Assurance

The Multi-Agent-System (MAS) at the Optimization and Network Control Layer is mainly performing service optimization and ensures that the deployed network service meets the respective service-level KPIs. In this section, we first provide the specific workflows for MAS deployment and MAS-based service operation [30] [31] [4]. Then, specific details of the security solutions designed are given. Related developments and implementation in terms of standards are available at [32] [33] [34] [35] [36].

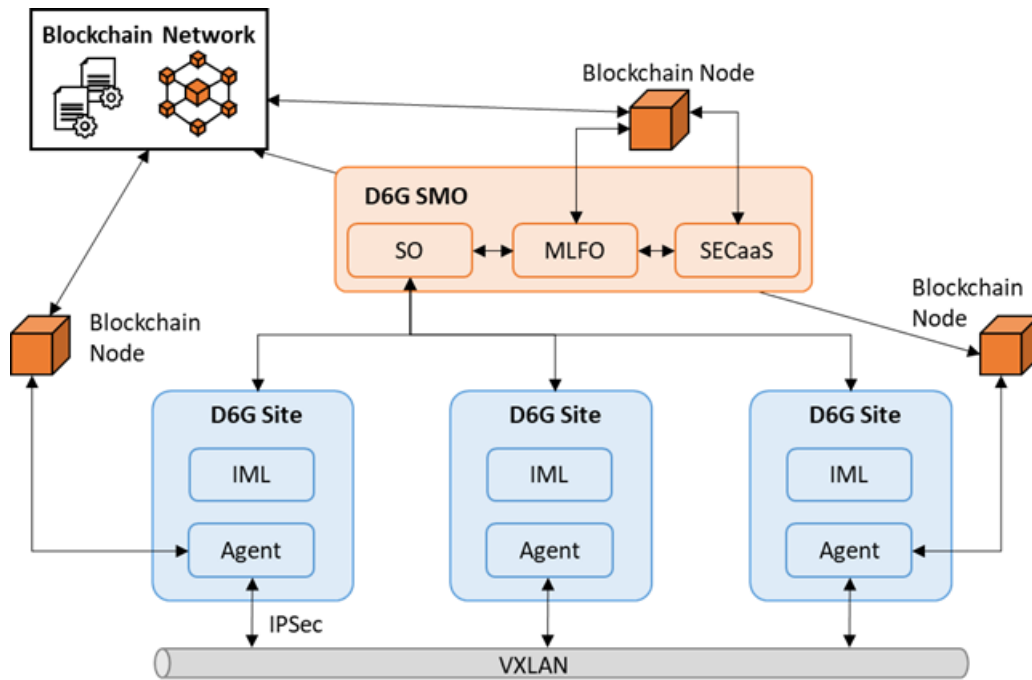


FIGURE 21 DESIRE6G ARCHITECTURE FOR SECURE DISTRIBUTED INTELLIGENCE.

3.1 MAS Deployment

Figure shows an overview of the essential DESIRE6G components required for executing the secure distributed intelligence. The architecture comprises multiple DESIRE6G sites in different geographical locations, each equipped with local networking, computing, and storage resources. Central to this architecture is the DESIRE6G SMO layer, which oversees the orchestration and lifecycle management aspects of the NS, providing guidelines for the agents and enabling them to operate autonomously. Within the SMO, the MLFO determines the deployment locations and connections of agents. A security-as-a-service (SECaaS) tool is used to improve the security of the agents by adding distributed attestation sub-routines for measurement and verification to their software. The SO is the responsible component

in the SMO to coordinate the deployment, where the IML is in charge of the interactions with the local virtual infrastructure.

A blockchain infrastructure is also part of the DESIRE6G solution that sets up two distinct SCs. The first SC dynamically manages the measurement and verification functions of agents, enabling mutual remote attestation mechanisms between agents. This allows each agent to check the integrity of another agent in the system, avoiding a centralized, Deny-of-Service-vulnerable verifier. The second SC ensures a secure exchange of keys/secrets for inter-agent communications. It is worth mentioning that DESIRE6G architecture offers a multi-use case blockchain framework, which, besides securing the MAS, also supports the federation of services.

Overview of MAS Deployment

Figure 22 outlines the proposed workflow for the MAS Pipeline deployment, which includes the MAS agents, their initial mutual attestation process, and their connectivity. This workflow consists of four phases: (i) deployment of the MAS pipeline; (ii) configuration of protected agents through a SECaaS binary hardening tool; (iii) mutual attestation; and (iv) secret/key exchange. The workflow is initiated by the SO during the deployment of the NS (see step 0 in Figure 22).

The SO starts with deploying the MAS pipeline and requests the computation of the topology of a MAS pipeline given the relevant details of the NS, i.e., the location of the VNFs, the performance required for that service, etc. (step 1). The MLFO computes the optimal MAS pipeline and returns a descriptor that specifies the agents' deployment location, booting image, and connectivity. The SO then interacts with the IML components at different DESIRE6G sites to deploy each agent (step 2). Inter-agent connectivity based on a virtual extensible local area network (VXLAN) that connects the D6G sites is also deployed in this step, even though it is not shown in Figure 22.

Once the agents are running and connectivity is established, the configuration of the agents starts (step 3). In this phase, the SECaaS system processes the native agent software (e.g., x86) and improves its security properties by including measurement, verification, and communication subroutines. Through this *wrapping* technique, SECaaS generates a reference measurement of the agent and stores it in a secure repository, ensuring its integrity for future verifications (step 4). The MLFO then deploys the wrapped images of the agents in the sites (step 5). Next, it creates smart contracts that will be used to record the state of agents during the remote attestation workflow and store the secrets/keys for the

MAS pipeline (step 6). It also creates blockchain accounts for the agents, including a public address and private key, deploys smart contracts to the blockchain network, and registers the agents' addresses (step 7). This allows the MLFO to control access to the SCs and add or revoke permissions in case of MAS pipeline reconfiguration or detection of infected agents. Once the addresses are registered, the MLFO distributes the blockchain credentials to the agents involved in the MAS pipeline for interaction with the SCs (step 8).

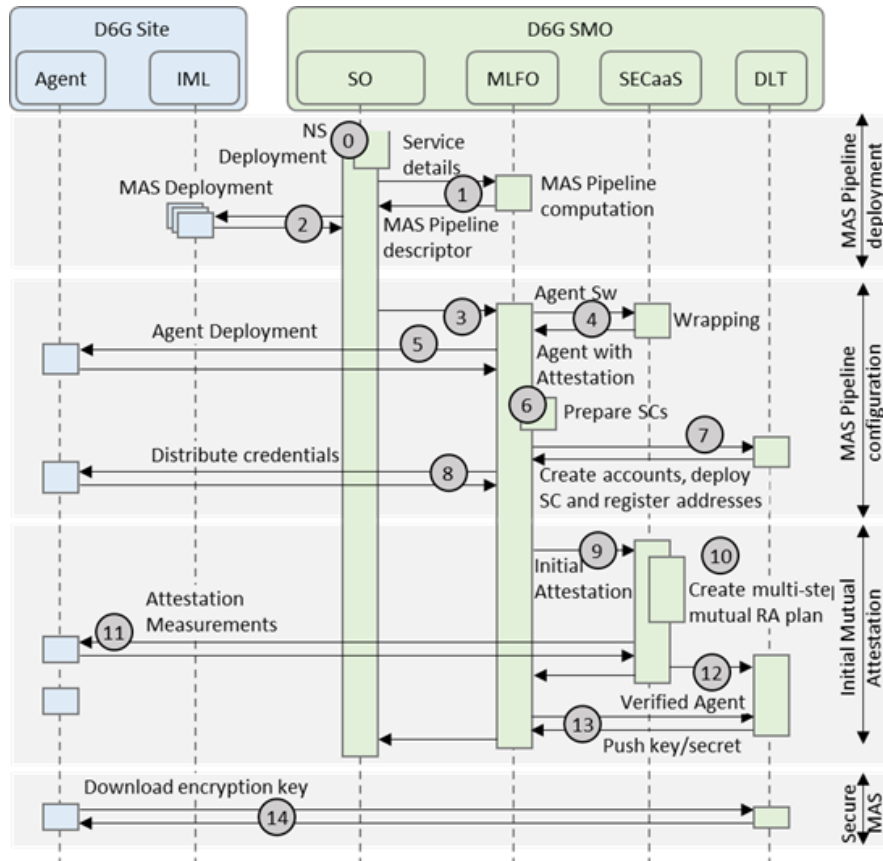


FIGURE 22 MAS PIPELINE DEPLOYMENT WITH INITIAL MUTUAL ATTESTATION.

To face group attacks, where a corrupted agent verifies another corrupted agent, an initial *root of trust* (RoT) is established for the first agent verification. To that end, the first attestation is triggered by the SECaaS when the MLFO confirms that the MAS configuration has finished (step 9). For the initial attestation, SECaaS prepares a remote attestation plan and asks the agents to run their attestation sub-routines for measurement and verification (step 10). The resulting hash value from the measurements (step 11) is compared to the one stored in the SECaaS repository and if it matches the agent is verified. A transaction is created in the blockchain for each remote attestation, including the test result, the identifiers of the involved agents (i.e., measured and verifier ID), a timestamp, and the updated RoT (i.e.,

last verified agent) in the first smart contract (step 12). Once the NS operation phase starts, the SC automatically manages the remote attestation process through an internal function that receives a group attestation scheduling plan (the sequence of agents for verification) and notifies the RoT with the reference measurement of the agent, without compromising its confidentiality to other potentially infected agents. These steps are performed periodically throughout the NS lifecycle. Finally, the MLFO pushes to the second SC the key/secret that the agents need to use to encrypt the inter-agent communication through the VXLAN tunnel (steps 13-14).

3.2 MAS-based Service Operation

Near real-time autonomous operation, where the actions are performed in the seconds scale, is a special type of network automation that requires the availability of new technologies and architectures to be developed in the control and data planes. One of the required technologies is pervasive in-band network telemetry (INT) which is used to get accurate measurements of relevant metrics, including QoS, e.g., delay and/or jitter, of the services supported by the network. Recent progress on hierarchical telemetry architectures with distributed intelligence showed the great potential of using such telemetry data for efficient network diagnosis and decision making.

In this section, we describe dynamic flow routing, as a use case of nRT network operation. Dynamic flow routing requires interaction between telemetry agents and flow agents in charge of making routing decisions. Among different routing policies, *multi-path* routing introduces flexibility in the design and operation of the network by allowing operators to split traffic demands into multiple streams that are routed independently of each other to the destination. Moreover, routes might have different utilization costs, and hence, the percentage of traffic sent through each route is a complex decision that needs to be dynamically tuned in order to meet robust QoS performance with overall minimum cost. Precisely because of the nRT decision making required, we developed a DRL-based agent that decides the routing policy to be applied based on traffic measurements and the end-to-end delay observed.

Proposed Workflow

For illustrative purposes, Figure 23 shows an example where several traffic flows (F_i to F_k), each following a multi-path routing strategy, enter and leave a network at different packet nodes. In the example, traffic flow F_i (from R_1 to R_5) can follow three different routes with different costs, where p_1 and p_2 are multi-

hop paths on the packet network, whilst p_3 uses an optical bypass connecting $R1$ and $R5$ through the underlying optical network.

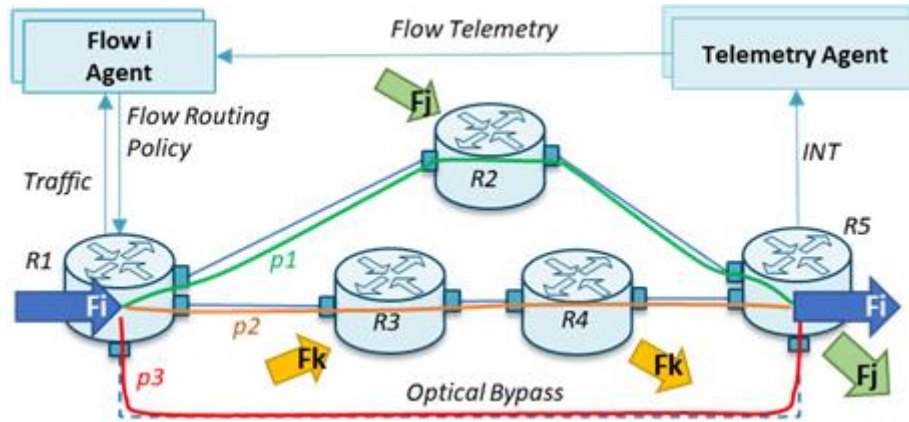


FIGURE 23 EXAMPLE: DYNAMIC FLOW ROUTING.

We assume that traffic flows are *splitable*, i.e., they consist of many sub-flows that can be routed independently. The objective is to find the flow routing policy that balances the incoming traffic of the flows among the available paths, so to ensure per-flow QoS, specifically E2E delay. Such a routing policy varies as a function of the incoming traffic of the flow and the network conditions, i.e., the traffic of the rest of the traffic flows in the network. Therefore, the routing policy decision making process is continuously carried out based on the incoming traffic and the E2E delay measurements that allow evaluating the quality of the decision making. Note that the state of the network is known, and it is indirectly represented by E2E delay measurements for the traffic flow. In this demonstration, we rely on the INT functionality provided by the P4 switches to measure packet delay. Specifically, a P4 collector collects, aggregates, and provides statistics of the delay measured by the switches supporting the traffic flows. Once the P4 collector pre-processes the QoS measurements, they are sent to a telemetry agent, which is in charge of producing flow telemetry statistics that are sent to the flow agent deployed at the source node, where flow routing policy decisions are made.

The distributed system consists of several interconnected software components. Specifically, the *telemetry agent*, which is part of the distributed telemetry architecture, includes: (i) a telemetry collector that periodically receives telemetry data from the P4 collector; and (ii) a per-flow telemetry processor. The role of these telemetry components is to compute the required measurements and statistics that characterize the current QoS of the traffic flow, from the received INT telemetry. In addition, *flow agents* are grouped in a single module per location named *multi-flow agent*. For instance, if several flows F_i (f_1 and f_2) enter router $R1$, the multi-flow agent at $R1$ would include flow agents for both f_1 and f_2 . Note that

one single flow agent makes routing decisions for each traffic flow. The *manager* running inside multi-flow agents has the role of collecting and distributing flow telemetry data and local input traffic data to the flow agents, as well as pushing the flow routing policies computed by flow agents to the P4 switch. Interfaces between agents and P4 systems are required.

The workflow for traffic flows $f1$ and $f2$ is outlined in 24. Let us assume that flow routing policies are configured at switch R1 (see step 0 in 24) to enable multi-path routing. Traffic input flow measurements are collected and sent periodically to the local multi-flow agent for every traffic flow entering in the network (step 1); the messages need to contain both packet and bit count for every traffic flow. Packet delay measured along the path is reported through INT messages to the P4 collector (step 2), which performs aggregations and computes statistics, e.g., minimum, average, and maximum of delay and jitter. Periodically, the P4 collector sends the computed delay statistics to the collector in the local telemetry agent (step 3). In addition, delay statistics are reported to a centralized telemetry system that stores them in a time series database (not shown in the workflow in 24). The aggregated QoS telemetry message includes per-path statistics for every flow. The received statistics are then processed by each of the flow processors (step 4) that computes per-flow QoS telemetry measurements and send them to the corresponding flow agent (step 5). When flow QoS statistics are received in the multi-flow agent, the manager triggers its analysis by pushing both traffic and QoS measurements to the corresponding flow agent (step 6).

An AI/ML algorithm, e.g., based on DRL, makes routing decisions, with the operational objective of guaranteeing that the E2E delay does not exceed a configured threshold (i.e., QoS requirement), while minimizing the use of the optical bypass. Routing decisions are gathered by the manager that eventually sends them to the P4 switch (step 8). A routing policy is defined as the percentages of input traffic to be routed through each of the routes. Internally, the P4 switch translates this policy into flow rules to efficiently perform packet forwarding according to defined percentages.

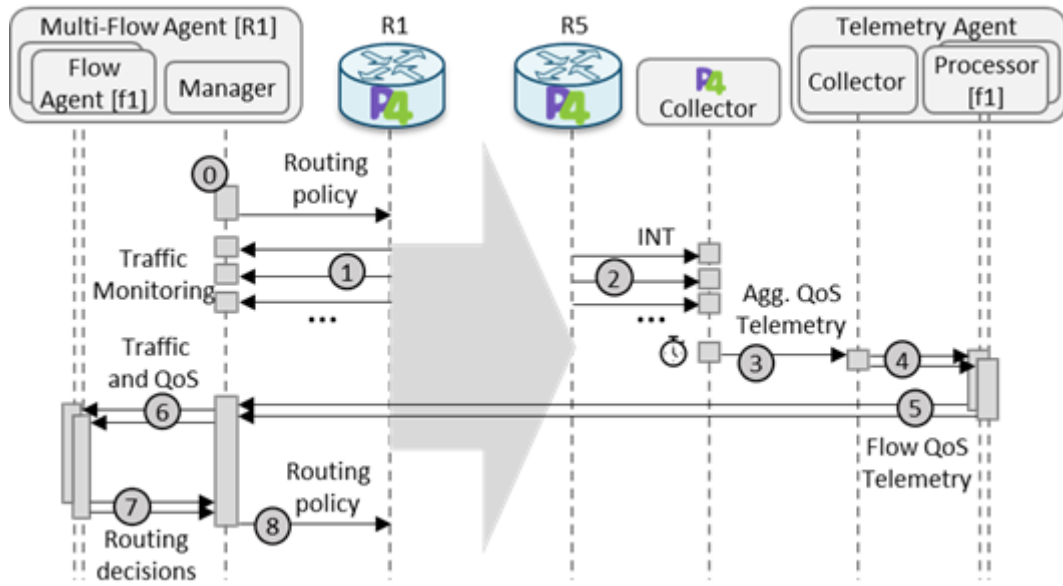


FIGURE 24 NEAR-REAL-TIME OPERATION WORKFLOW.

3.3 MAS-based Service Reconfiguration

In this section, we concentrate on mobility scenarios in E2E NSs supported by RAN and packet-optical transport network segments. Here, it is important to anticipate NS reconfiguration to guarantee the committed performance. To that end, a special procedure is proposed to ensure that the MAS pipeline is ready to operate in short time. The procedure includes DRL-based traffic prediction models for the NS, which are trained by agents in the MAS. Such prediction model is shared and used by other agents to probe the resources allocated during the NS reconfiguration. Obtained results help to select the right pretrained model to be used for operation after the NS reconfiguration

Figure illustrates the targeted mobility scenario, where the distributed MAS-based control reconfigures the NS and its related MAS pipeline in advance for service assurance. The top part of Figure reproduces the resources allocated to the NS connecting a drone, currently being served through RAN A1, to an application running in an edge/metro DC facility. A MAS pipeline oversees the service and guarantees that the delay in the transport network does not exceed some maximum value (denoted d_{max}).

In the following, we concentrate on the actions made to guarantee d_{max} . Specifically, an agent (Agent 1) collocated with ingress packet switch S1 applies routing policies to balance routing among a set of routes that have been previously allocated by an SDN controller (in the example, routes P1 and P2 are available). The resulting delay is measured by another agent (Agent 2) collocated with egress packet

switch (S3) that connects with the edge/metro DC. The MAS-based control can manage the delay component introduced by the transport network by balancing the traffic sent through the available routes. Let us assume that the drone follows a trajectory that separates it from A1 and makes it closer to A2. If handover is performed at that time, only best effort connectivity will be provided, which could impact service performance. To avoid that, resources need to be allocated before handover happens. The bottom part of Figure 25 shows that two new routes (P3 and P4) are allocated and a new agent (Agent 3) is deployed to control routing between packet switch S3 and the edge/metro DC.

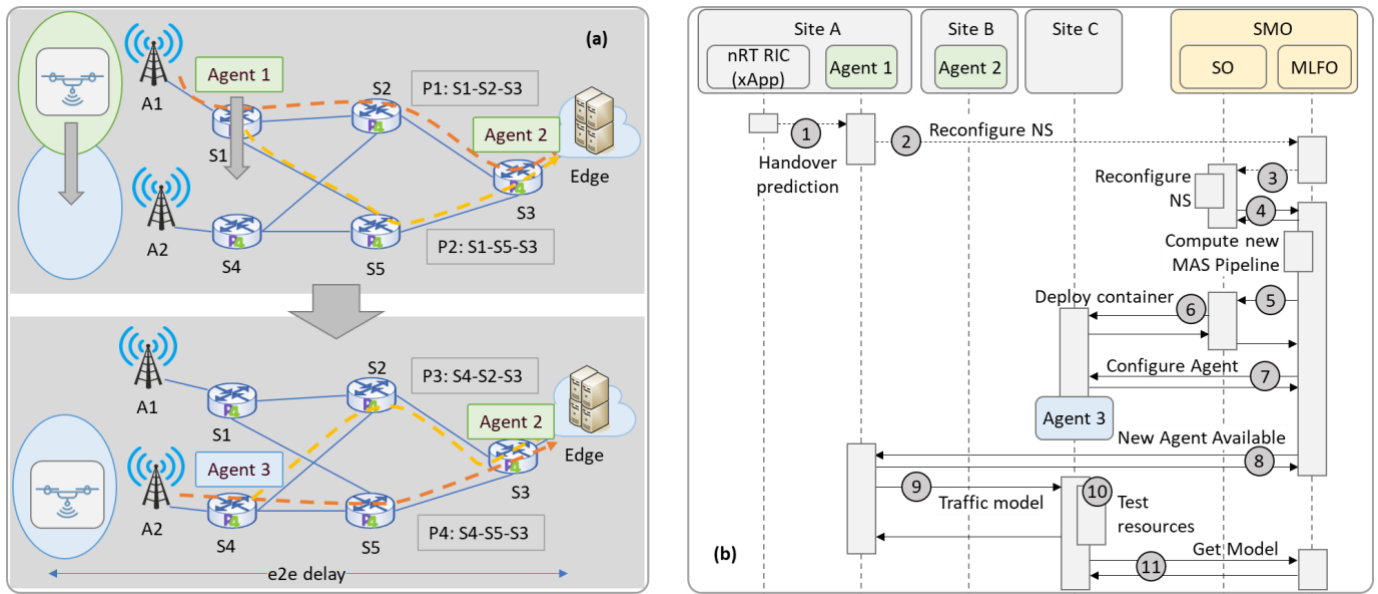


FIGURE 25 ILLUSTRATIVE MOBILITY SCENARIO (A) AND PROPOSED WORKFLOW (B).

To anticipate handover with enough time so resources can be allocated (e.g., 10s), an xApp running in the nRT RIC follows the trajectory of the drone and triggers the reconfiguration of the NS, including the related MAS pipeline. Figure 25b presents a simplified version of the proposed workflow. The xApp notifies the possible handover to the collocated Agent 1 in the MAS controlling the service (step 1 in Figure 25b), which in turn notifies the SO via the MLFO (2, 3). The SO finds the most appropriate forwarding graph for the NS including the predicted mobility and reconfigures the NS. In the example in Figure 25a, the SO requests the new connectivity to the SDN controller, which sets up paths P3 and P4 (details are omitted in the workflow for simplicity). Once the NS has been reconfigured, the SO requests the MLFO to reconfigure the MAS pipeline and gives the details of the new forwarding graph (4). The MLFO computes the new MAS pipeline, which includes new Agent 3 and requests the SO its deployment (5). The SO deploys the container for the new agent with the help of the local virtual infrastructure manager in Site C, as well as the MAS pipeline connectivity with the help of the SDN

controller (6). Then, the MLFO configures the new agent (7) that is ready to participate in the nRT control of the NS, which is communicated to the other agents in the MAS pipeline (8). Therefore, Agent 1 distributes the traffic model for the service to new Agent 3 (9), which uses it to select the RL model that needs to be used to operate the service. The procedure includes injecting traffic following the received traffic model through paths P3 and P4 to measure the delay (10). An algorithm takes such measurements and selects the model that better fits the new scenario from the once that are already pre-trained and available in the MLFO. Finally, Agent 3 requests that model to the MLFO and it is ready to operate (11).

Knowledge sharing between agents enables seamless network service (NS) reconfiguration. Specifically, Agent 1 shares with Agent 3 a traffic model for the NS, which has been trained in parallel to routing operation. Both traffic and routing operation models are based on Twin Delayed Deep Deterministic Policy Gradients (TD3). In the case of traffic prediction, the model predicts the NS traffic for a time window in advance. The state, reward, and actions are computed periodically. The state is a vector with the input traffic over the last period until t . Action is a vector with the minimum and maximum traffic expected during the period. The reward uses the maximum of differences between prediction and measured traffic (see eq. (1)).

$$reward = -n * \max(E_{min}, E_{max}) \quad (1)$$

The used notation is as follows: *i)* M : set of models, each characterized by $\langle d_{max}, P_{max}^i, P_{min}^i \rangle$ for every path considered by the model, where P_{max}^i, P_{min}^i are the percentages of traffic sent through P^i ; *ii)* S : set of probe scenarios, index s . Each scenario is characterized by $\{P_{t_0}^i\}$; *iii)* D : set of delay measurements, $D=\{d_s, \forall s \in S\}$; *iv)* n : maximum traffic of the flow; *v)* $Tr_{(t)}$: Actual min and max traffic for the window; *vi)* $Pr_{(t)}$: Predicted min and max traffic for the window; and *vii)* $E_{(t)}$: Absolute prediction error; $\text{abs}(Tr_{(t)} - Pr_{(t)})$. The traffic model is used for reproducing operation scenarios that will happen after the handover. To that end, the new agent uses the following to select a model able to ensure the required performance. The algorithm receives the set of models M , the set of probe scenarios S , and the set of delay measurements D . The models in M have been trained on specific topologies with unknown background traffic and validated in completely different topologies, which enables that the operation of each model can be characterized by attributes like delay and routing policies. The algorithm compares the delay measured under each scenario with d_{max} . If the maximum delay is guaranteed, the models that can operate in that probe scenario are selected as candidate models and labelled by distance to the measured delay. If

there are candidate models, the one with the lowest distance is selected; otherwise, no model is selected, which is notified to the SO via the MLFO to provide more resources for the NS.

3.4 MAS Security

The work produced on MAS agents' security is discussed in the following sub-sections, bringing progressive improvements for a scalable use of D-MUTRA, a novel blockchain-based, mutual application remote attestation service. Our work provides D-MUTRA with key differentiating features compared to the state of the art, making remote attestation zero-touch and penalty-free process where workloads can be deployed on any platform, without modification.

3.4.1 Security considerations of spontaneous attestation workflow

3.4.1.1 Problem statement

The spontaneous agent remote attestation, as defined previously, does not prevent the execution of tampered agents but detects their corrupted states. We have worked to define a practical solution which remediates this security pitfall, conditioning execution on a successful attestation verification only. The solution relies on an exchange of an execution token between the agent with the other components of D-MUTRA, once the attestation test has been successfully produced.

3.4.1.2 Agent-specific and one-time execution token mechanism

The delivery of the execution token shall prevent multiple use of the token by the same agent (which can still be modified between launches) or worse to start any corrupted agents. To meet these two assertions concurrently, the following techniques have been considered.

Mutual authentication by mTLS (prior to token delivery)

Mutual TLS (aka mTLS) leverages x509 certificates to authenticate both parties (i.e., client, server) before establishing a secured TLS channel. For the transmission of the agent execution token, mTLS would provide the assurance that the token is transferred by the server (i.e., potentially the D-MUTRA's SECaaS) to an authenticated destination (i.e., the agent acting as a client). However, this would not prevent the token duplication and transmission to another tampered agent, including itself. Therefore, the tokens must be bound to the agents, meaning be useful only for the corresponding agent that was fordr. Last,

mTLS requests a Certification Authority (CA) to deliver the client and server valid certificates that is not efficiently coping with dynamically generated agents. We have decided not to use this technique.

Agent-bound token

In the domain of Digital Right Management, activation tokens are generated by a central activation server (e.g., held by the DRM vendor) using sufficiently accurate (i.e., with sufficient unique platform identification ability) and perennial (i.e., long-lived) platform invariants (e.g., O.S partition, CPUID). The tokens produced by the activation server cannot be used elsewhere than on the platform they were produced for. At the start and during execution of the software, a software-appended DRM routine collects both the token and the local platform invariants again, checking their correlation to keep the execution. The activation token once collected and stored at the execution platform from the activation server can be used for any subsequent starts of the software on that platform. Last, by use of asymmetric encryption at the activation server, tokens cannot be forged elsewhere than in the activation server, holding the RSA private key. DRM solutions are exposed to tampering attacks, produced in the “privileged adversary threat model”, aka dynamic analysis and where the attacker maps and tampers the loaded software memory pages. DRM solutions deliver a relative security assurance which depends on how deep and buried the DRM primitive is, using code obfuscation and anti-tampering techniques. Typically, a naïve implementation terminates the correlation check as an easily **invertible** Boolean test.

One Time Token

Rarely used, One Time Tokens (OTT), triggers only one execution. Their generation needs to grasp the process status in its execution environment (e.g., process ID allocated by the O.S) which will differ with a sufficient probability at the next execution. OTT solutions require the transfer of this execution context metadata from the software to the server, the generation of OTT by the server and its transmission back to the software, at each launch of the software. Thereafter, the same type of execution control routine (as used in DRM above), appended in the software, checks both the correlation between the OTT and same execution contextual elements it collects again.

Again, OTT solutions only bring a relative security level as privileged adversaries with full access on the execution environment memory map can tamper the execution control routine, typically by inversion of the execution control routine test.

TABLE 3. SECURITY OF EXECUTION TOKEN COMPARISON

Considered solution	DRM, (perennial token)	OTT
Changes required on the software	Yes, the DRM routine is appended	Yes, the OTT routine is appended
Latency at start	A latency in the range of 3-5 sec is caused by the activation process with the remote server, which includes the forging of the token by the server. Once the activation has been carried out, the activation token resides on the platform, another recurrent latency (in the range of 100 msec) is caused by the DRM routine. This timing actually depends on the level of obfuscation of the DRM routine.	At each new launch of the software, a similar 3-5 sec latency is caused by the exchange with the OTT server (and the generation of the OTT). Latency can expand with the level of obfuscation of the execution control routine.
Assessment of the security level	<ul style="list-style-type: none"> - Full when attackers do not have administrative right on the platform - Relative with privileged attack threat model. The security level depends on the employed combination of code obfuscation and anti-tampering techniques 	<ul style="list-style-type: none"> - Full when attackers do not have administrative right on the platform - Relative with privileged attack threat model. The security level depends on the employed combination of code obfuscation and anti-tampering techniques

3.4.1.3 Practical implementation in D-MUTRA

As shown above, execution token does not bring more than a relative security level in high adversarial conditions (i.e., privileged attacker practicing dynamic analysis). The security level is driven by the employed obfuscation techniques which create opacity on tamper-trapped execution control routine. There is no plain solution against such adversarial threat model, except by leveraging trusted execution environment which is out of scope here. With this fact in mind, we have decided to consider a baseline approach, without employing complex anti-tampering and obfuscation techniques for the following reasons:

- Low plausibility of the privileged attacker threat model in networking, where platforms are not own by users.
- Severe latency impact caused by anti-tampering and obfuscation techniques
- Severe latency caused by OTT full cycle.

Our token solution features the following:

Agent discriminating factor:

- A unique **Agent ID** (large integer) is written into the agent's ELF .text section by the **SECaaS**.
Duration: **~1-5 ms** (offline, negligible at runtime)
- This ID is generated **at the time of reference measurement** creation by the SECaaS.

Duration: **~1-2 ms** (simple ID generation and storage)

Token composition:

- Certificate of the Agent ID, signed by the SECaaS
Duration: **~5-10 ms** (asymmetric signing, e.g., ECDSA or RSA-2048), performed offline during attestation, no runtime cost

Token verification routine (i.e., embedded into agents)

- Check the authenticity of the token and verify simultaneously the correlation with the locally resident Agent-ID , (i.e., produces the hash of the Agent ID, decrypts the token and verify identity)
- Once the verification is carried out, the routine jumps to the entry point of the program to start its execution.

The total time for the token verification routine is approximately **80 milliseconds**, including all steps from token decryption to program launch.

Security threat models:

- Tokens cannot be used by another agent.
- Tokens can be used for previously attested tampered agents. However, it is worth noting that our runtime integrity verification will detect the tampering during execution.

3.4.2 Operational considerations. Fixing latency at start

3.4.2.1 Problem statement

Discussions with UPC have led to reconsider the remote attestation workflow significantly. MAS agents shall start their execution instantly and remote attestation by D-MUTRA (i.e., about 2 seconds for the full cycle of remote attestation) does not comply with ultra short latency at start. In fact, we believe that ultra short latency at start is a common trait of all advanced 6G services.

A detailed breakdown reveals that **around 90% of this latency originates from the use of the blockchain**, specifically the **event-based interaction model in Ethereum**. The blockchain is used to store and validate attestation evidence, but the confirmation of transactions combined with the asynchronous event polling mechanism introduces substantial delays:

- Blockchain interaction and event confirmation: ~1,800 ms
- Measurement collection and request preparation: ~100–200 ms
- Trust decision and response: ~100–200 ms

3.4.2.2 Continuous attestation

Continuous attestation refers to the periodic integrity measurements produced during the code execution. It delivers a similar integrity assurance as offered TEE but with lesser operational complications, memory and processor resource consumption and finally novel forms of attacks exploiting the TEE (i.e., evil TCB where a vulnerability can be hiddenly exploited, DoS attack exploiting TEE hard lock integrity policy). Runtime integrity verification uses the same method and verified element (i.e., the memory mapped text section of the workload ELF format) used by our remote attestation. The frequency of the measurements is directly correlated to the workload performance penalty.

Our implementation sets up a secondary thread which produces the measurement. This thread operates independently from the workload it measures. It shares the process memory, hence enabling its access and measurement and, processor resources. We had been considering three different techniques exposed here:

- **Spread-over-time measurement:** The technique consists in spreading the hash calculation over time to limit the penalty as it consumes less processor resources per unit of time. One hash is

produced by one loop collected fragments of the data blob and xoring it with the previous iteration outcome. Our technique expands the time to produce the hashing by adjusting the time of processing of each iteration. As our technique chains measurements one after the other and as each measurement takes an adjustable duration, our technique adjusts the number of produced measures per unit of time. We adjust the time taken by one complete measurement using system call sleep command into the main xor loop and assess the performance with timestamps. Our tests and measures revealed incoherent results, sometimes going in the unexpected way (e.g., more frequent measures coming with an acceleration of the measured process). Our vision is that we had fallen into the opacity of the process optimization done by the processor which dynamically reallocates resources between the different processes. Hence, congesting a shared resource at a given time may lead to a reallocation of the process resource, reshuffling the cards but without our knowledge. The processor optimization is multi-factor, depends on all other processes and leads to sudden non-linearities. (e.g., sudden relaxation of resource constraint on one processing node). Except in situations where all processing nodes are close to their saturation level, we can predict with a sufficient likelihood the benefit when relaxing of processing constraint on one node. However, we are still looking at this technique, removing ambiguity on the ability or inability of the optimization policy to separate the measured process from its attached measuring thread. This work is motivated by the fact that this technique does not require administrative rights on the platform which scales its usage, typically in off-premises cloud operations.

- **Linux kernel cgroups V2 feature: Cgroups** enables to allocate a percentage of CPU consumption to different process ID. Hence, the consumed resources by the measuring thread can be adjusted to limit the penalty induced on the measured workload. Compared to our process time spreading programmatic approach as stated above, cgroups is a system function which directly sets the rule for the system, and the rule is simple (i.e., affect no more than x% of the processor resource at any given time to that process). Our measures, based on different CPU %s (i.e., cgroup settings) have shown coherent results aligned with our expectations. However, cgroup command requires root privilege by default. Administration delegation can be setup in modern Linux distributions, enabling a user to adjust cgroup policy. Moreover, VM and containers

do not enable cgroup leverage uniformly. We are analysing the operational and business conditions to exploit cgroup resource management by cloud clients.

- **On-demand integrity verification which** can be triggered through D-MUTRA, reducing to nil the impact on performance as one measurement only is triggered. This approach, illustrated in Figure 26 is on-going and has not yet been implemented. To do so, we will command the measuring thread with an external trigger. How design leverages SIGSTOP/SIGCONT commands through a secure API embedded into the same user ID.

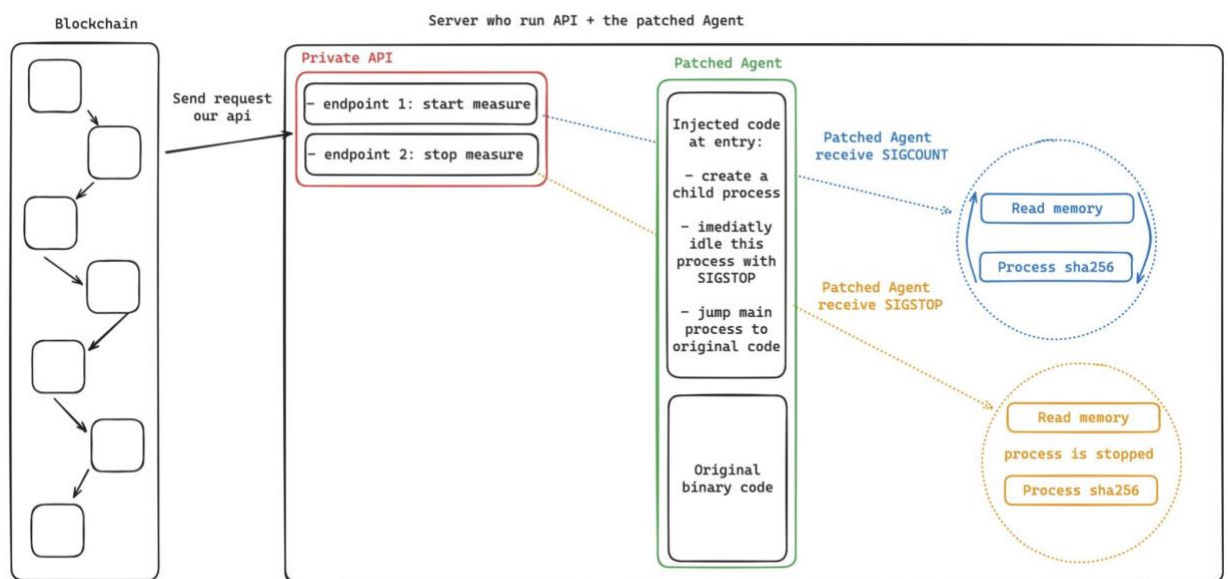


FIGURE 26. ON-DEMAND INTEGRITY MEASUREMENT IMPLEMENTATION

TABLE 4 COMPARISON OF TECHNIQUES FOR CONTINUOUS ATTESTATION

Criteria	Spread over time hashing	Cgroups	On-demand
Adjustable overhead	Yes, by modifying the sleep function argument (i.e., pause time before resuming)	Yes, the cgroup argument can be	NA
Administrative rights required	No	By default. Possible activation w/o administrative	No

rights, leveraging User namespaces or through administrative delegation.
--

3.4.2.3 Attest after starting security pattern

To eliminate latency at service start, we leverage runtime integrity verification. Doing the integrity verification at runtime relaxes the workload start latency constraint plainly, as the workload starts its execution before remote attestation is worked out. Security purists could claim that this pattern creates a temporal blind zone and breaks the security posture which imposes that no code can execute before being remotely attested. This claim although correct can be countered with the following arguments:

- Code authentication, an easy zero-touch process (as all elements are delivered in the workload manifest) enables to check at the execution platform that the code is integrated at onboarding. The induced latency is in the range of 1 msec, hence complying with urLLC service instantiation. Once authentication check has been done, our continuous runtime integrity takes over, bringing continuous integrity assurance. Combining both methods of authentication and runtime integrity verification closes the security blind zone.
- If the above-stated combination cannot be implemented, the first integrity verification will generate a blockchain state within less than one second after the workload starts, hence producing a tampering alert in near real time. The security blind zone shall be put in the general current context where remote attestation is not commonly practiced as it engenders complex operations.
- With low occurrences of tampered workloads, most of the workloads will be integrated during both onboarding and during execution. Our solution avoids penalizing all workloads with a start-up latency, accepting the risk induced by short temporal integrity blind zone of the very few corrupted workloads.

3.4.3 Novel drop-off zero-touch workflow

3.4.3.1 Problem statement

D-MUTRA workflow includes a pre-deployment stage where workloads are measured (for reference measurement creation) after having been themselves appended with Prove, Verify and DLT routines.

These operations were realized by the SECaaS. We believe that this pre-deployment workflow is hardly acceptable as we modify the client workload first and deviate the clients' usual workflow.

3.4.3.2 Novel security scheme.

Provided that the workload is containerized, a novel and highly simplified workflow has been designed, cancelling any modification to the workload. By use of Docker compose facility, we associate untouched workloads with a pre-defined sidecar container which contains the Prove, Verify and DLT routines. To collect the reference measurement, the sidecar produces it as soon as the workload starts to execute for the first time.

As the payload could have been tampered with prior to the deployment, we consider the combination with an authentication process as stated above. Workloads can be just dropped off and be automatically verified during their execution by the sidecar container, with no more SECaaS stage. This novel simplified workflow is depicted in Figure 27.

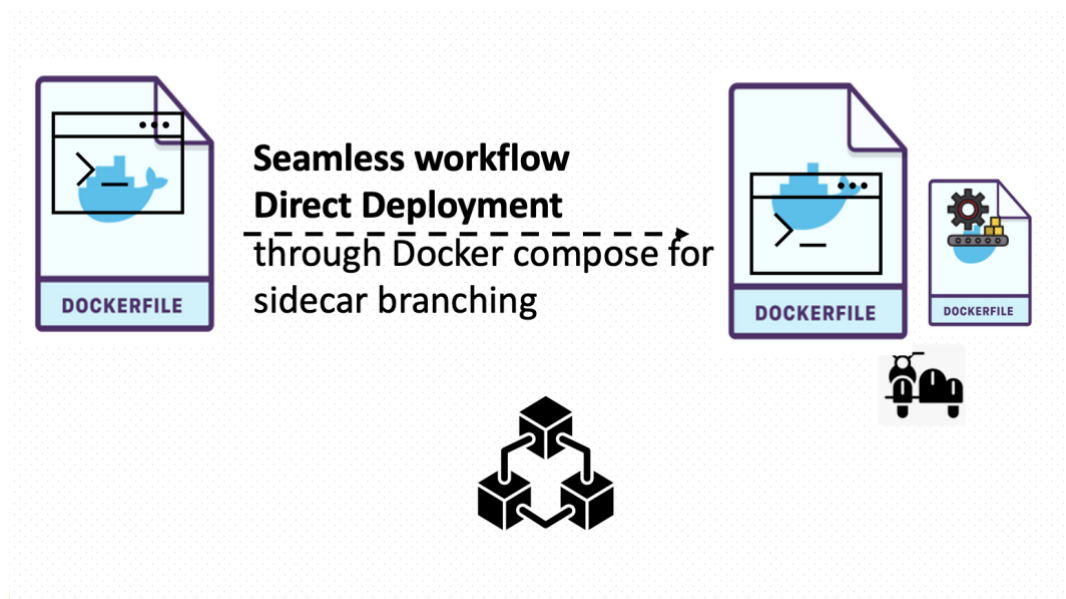


FIGURE 27 SECAAS-FREE DOCKER COMPOSE BASED WORKLOAD DROP-OFF WORKFLOW

3.4.4 Blockchain considerations

3.4.4.1 From Ethereum PoS to hyperledger fabric

For the sake of sustainability and more specifically in terms of memory and processing resources, we are considering using hyperledger fabric instead of Ethereum in its permissioned, private and proof of stake consensus flavor. The foreseen benefits can be summarized as below:

Hyperledger Fabric consumes very little energy thanks to its lightweight consensus mechanism, unlike more resource-intensive models such as proof-of-stake. Unlike Ethereum, Hyperledger Fabric does not require each node to constantly run a virtual machine (such as the EVM). It also allows the use of different databases (e.g., CouchDB, LevelDB) and supports pluggable consensus mechanisms (such as Raft), enabling lightweight deployments tailored to actual use cases.

3.4.4.2 Limitation of Blob inflation rate

In contrast to remote attestation done only at onboarding, runtime integrity verifications, produced periodically, will quickly overload the blockchain if a transaction shall be produced at each measurement. For instance, storing one blockchain transaction every two seconds over a year results in approximately **7.9 GB of data**.

Two alternatives are considered to limit the blockchain inflation rate. First, integrity measurements will be stored off-chain using secure logging mechanisms such as append-only databases, decentralized storage (e.g., IPFS), or auditable Merkle tree-based systems. This approach ensures data traceability and verifiability without inflating the blockchain. Second, we will only trigger an on-chain transaction in case of detected tampering – for instance, when a mismatch occurs between a current measurement and a reference state. This event-driven model drastically reduces the number of blockchain writes while preserving trust and auditability for critical security events.

3.4.5 SECaaS hardening support to trusted execution

3.4.5.1 SGX implementation

Solidshield's Systemic software security solution is a SECaaS modeled solution which hardens x86 software workloads (e.g., VNF) against confidentiality and integrity attacks. Systemic brings two different

flavors for software deployment w/ or w/o Intel's SGX enablement. When SGX is leveraged, the Systemic hardening routine itself (and alone) is set up inside SGX enclave, as opposed to a complete workload. In other words, the workload (e.g., VNF) is hardened with an appended Systemic routine which is itself placed into the enclave. The rest of the workload stays outside of the enclave. The core merits of this design are listed below:

- The Systemic routine brings security to the workload (e.g., VNF) with different functions. Typically, it assures that the VNF is untampered. This posture exposes it to attacks. Only SGX full confidentiality and integrity attributes ensure that the routine is totally safe. Conversely, when SGX is not available, code obfuscation and anti-tampering techniques shall be applied to prevent easy tweaks, breaking its delivered security, but without delivering plain security attributes against both confidentiality and integrity attacks. Beyond this extension of security of Systemic routine with its placement inside SGX enclave, another benefit is the performance gain of the routine compared to a what elevated code obfuscation and anti-tampering techniques would generate.
- All preparative operations for SGX enablement are done once as the routine is common to all uses. If the complete VNF had to integrate the enclave, an ad hoc preparation stage would be required (e.g., remove system calls or non-supported x86 instructions). SGX was known to be uneasy to exploit with a minimal developer's time investment and training effort level to get started, but this scheme removes any difficulty on the VNF developers' side as SGX is only used by Systemics's routine.
- The SGX license by Intel has been delivered once for Systemic and the associated delivery of a whitelisted keypair delivered to Solidshield. At that time, the workflow would have been far more perilous if the complete VNF had to embed the SGX. In June 2022, Intel has removed the license requirement and the delivery of whitelisted key pairs.

This restricted implementation has led to a simple workflow for Solidshield's clients which only had to select the output of the SECaaS, ticking SGX enablement as show in Figure 28.

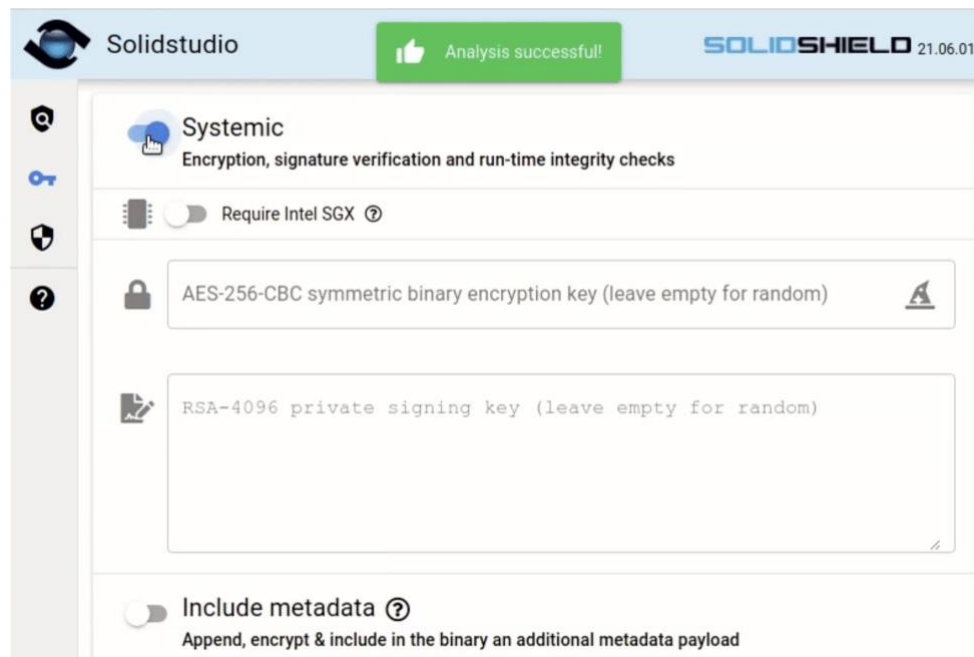


FIGURE 28 . SYSTEMIC USER ACTIVATION OF SGX

In Figure 29, a performance assessment has been made to assess the real costs of SGX placement as shown in Systemic SGX performance (for a symbol resolution) is very close to unobfuscated Systemic (aka Dynamic Systemic) and significantly higher than Systemic obfuscated flavor (ie, aka Systemic Full).

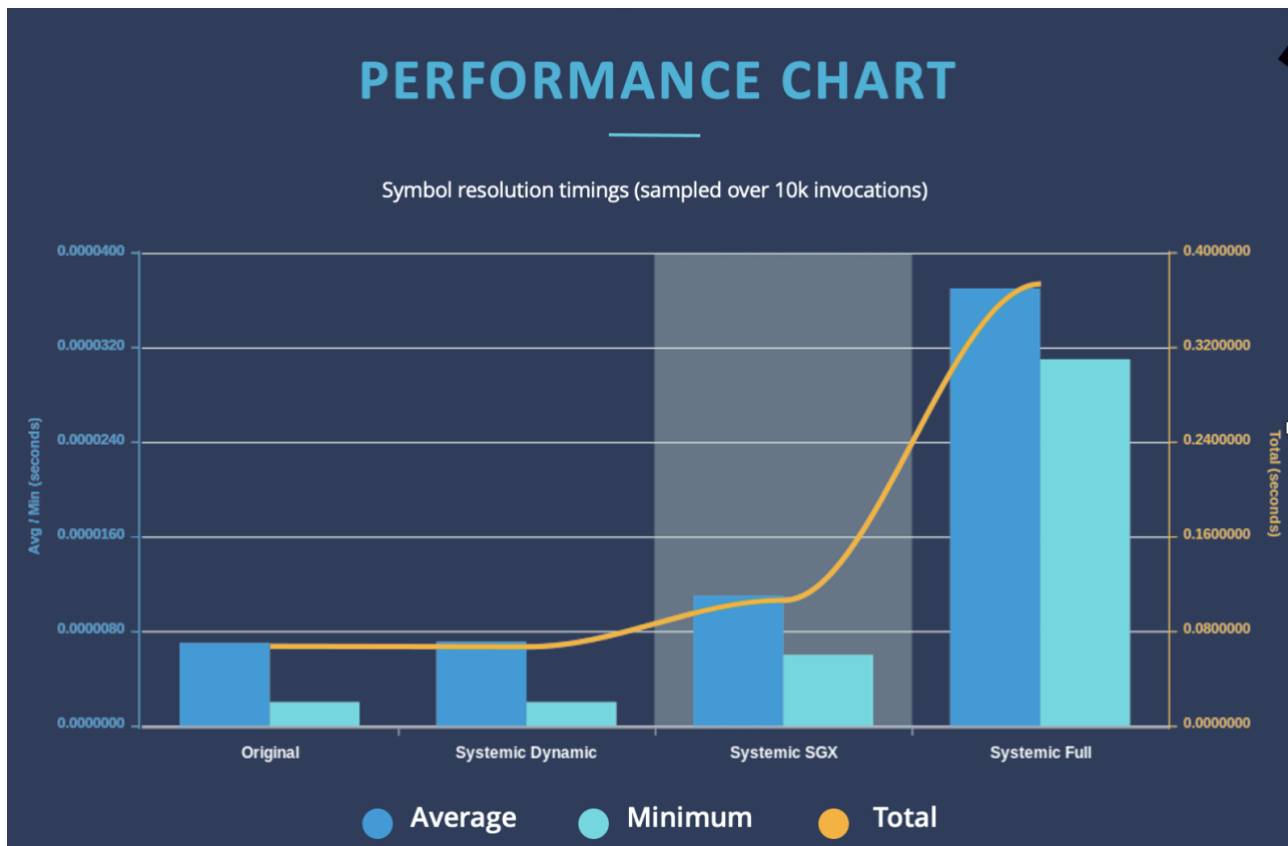


FIGURE 29. PERFORMANCE PENALTY INDUCED BY SGX ON SYSTEMIC

As discussed above, setting up SGX flavor was an easy operation for Solidshield's clients. However, the main difficulty relates to the verification of all provisions to get SGX enabled on the execution targets. A check of the CPU (ie, retrieving first the CPUID flag) was to be done to verify if the CPU came in Intel Core 6th to 10th gen (client) and Xeon E3/E, Scalable Xeons Intel Core 6th to 10th gen (client) and Xeon E3/E, Scalable Xeons series. Moreover, SGX must be enabled in BIOS/UEFI setup while the operating system must also be SGX enabled with the required driver (e.g., Windows 10/11, Linux kernel 5.11+). Without stating that these deployment conditions are hardly attained, they certainly hinder the deployment of VNF or any forms of software, notably in cloud operations or off-premises.

3.4.5.2 SGX (partial) deprecation and replacement

Intel had deprecated SGX technology (in terms of lack of support) according to two classes of clients. Owners of "clients" machines (i.e., P.C owners) from the 11th (aka Rocket Lake), officially classifying it as "deprecated" in 2022. The shift was widely reported in January 2022, notably affecting Rocket Lake (i.e., Intel x86 processor 11th gen) and Alder Lake (i.e., Intel's x86 processor 12th gen) desktop.

For server clients, SGX is still holding to date. It remains supported and continues being developed for cloud and enterprise use. Intel community moderators and sources confirm “no plans to deprecate SGX on supported Intel® Xeon® Scalable processors” as stated in⁷.

Additionally, in terms of attestation services, Intel has officially announced the end-of-life for the Intel® SGX Attestation Service (IAS)—the EPID-based remote attestation—on April 2, 2025. Access to the IAS development environment was restricted starting September 29, 2024, as. Enclave activation must go through client implemented and distributed activation services (as opposed to IAS centralized service) leveraging the DCAP/EDCDSA attestation stack. This move was motivated and synchronized with the suppression of Intel centralized whitelisted key pair delivery. In the novel DCAP/EDCDSA schemes, clients have the autonomy to sign their enclave using a key generated by the enclave itself and using a signed certificate produced by Intel’s SGX attestation ledger. All operations are local.

These decisions by Intel were synchronized with Intel’s TDX, launched in mid 2021, in a marketing move towards AMD’s SEV and ARM’s CCA technologies, offering transparent setup for complete VM for cloud operations. Although TDX is destined to cloud operations at the first place, it is considered by the community as the replacing technology for SGX. In practice, TDX leverages SGX if remote attestation is required and constructs a specific packaging, drastically simplifying the workflow.

3.4.5.3 SECaaS support of novel forms of TEEs

Intel’s TDX, AMD’s SEV-SNP and ARM’s CCA are the novel forms of TEE, aligned on their marketing promises of simplified usage. They are totally aligned with the novel offering of cloud vendors for confidential computing. The simplification of the workflow does not come for free as discussed in [37]. Serious risks of uncontrolled memory consumption, excessive overhead and novel threat vectors emerge with these technologies, breaking the restricted trusted computed basis, the foundation of SGX. Installing a complete VM inside a TEE consists in inflating drastically the trusted computed basis. The associated risks expand while user awareness diminishes. Evil TCB risk augments with the size of the TCB when a

⁷ community.intel.com+2community.intel.com+2urtech.ca+2.

large size VM including potentially vulnerable operating system and applications are present. Potential vulnerabilities are not mitigated by the placement inside an enclave, hence potentially exploitable even though the attacker must operate without visibility (i.e., confidentiality assurance brought by TEE). A second less discussed but potentially severe derives from the hard-lock integrity policy of TEE, with a brutal DoS threat model without pre-knowledge of the victim code, accentuating its plausibility.

Motivation to leverage novel forms (i.e., large TCB) TEEs)

First, the paradigm shift caused by the TCB size augmentation associated with the suppression of any preparatory work brought by large TCB TEEs raises the question of the relevance of our hybrid TEE use as designed for SGX, conceived to deliver a seamless setup workflow for application vendors. To make it simple, there is no more motivation in a dual execution environment setup with the application outside the TEE and our appended routine inside the TEE. In fact, the split will not ease the setup while it induces performance penalty caused by the enclave calls aka ECALLS and OCALLS in SGX terminology as well as marshalling the data coming from the outside. In short, our hybrid implementation is irrelevant with large TCB TEEs, and the complete content of the protected code must be part of the TCB.

In this direction, another question is what extra security is brought by SECaaS hardening of a code that will be embarked in a TEE? At first sight, the security gains are inexistant. As the TEE confidentiality and integrity assurances are plain and total (i.e., if we exclude side channel attacks from the threat model), our hardening of the code is superfluous and CPU excessive. Looking more closely, when considering the insider threat, more plausible with large TCB and when no sanitization prior on-boarding is worked out by users, novel subsidiary and secondary security needs may emerge. Process isolation, process integrity monitoring and process performance monitoring are still relevant. In fact, all SECaaS hardening functions are relevant when considering the insider threat model.

Integration works for SECaaS hardening inside large TCB-TEE

We had enumerated the several pre-conditions for leveraging Intel's TDX and AMD's SEV-SNP large TCB TEEs in Table 5. In addition, the last line relates to the main cloud providers packaged offers (i.e., confidential computing). Looking at the table, the DevSecOps operators will certainly appreciate the first and the last lines which make things easy but will raise more concerns on the other lines. Packaged offers from cloud vendors deliver ready to use TDX or SEV-SNP enabled platforms, where someone simply sets its VMs upon. This removes risks associated with the technical pre-conditions which must be considered for someone making its own implementation.

Feature	Intel TDX	AMD SEV (including SEV-ES/SEV-SNP)
Need for application changes	No	No
Processor Type	Intel Xeon Scalable Processors, 4th Gen Sapphire Rapids or newer	AMD EPYC Processors, 2nd Gen (Rome) and newer (preferred SEV-SNP on 3rd/4th Gen Milan/Genoa)
BIOS/Firmware Requirements	UEFI BIOS with TDX and relevant Intel TXT (Trusted Execution Technology) enabled; requires Intel ME firmware support	UEFI BIOS with SEV/SEV-ES/SEV-SNP enabled; requires PSP (Platform Security Processor) firmware
Operating System Requirements	Linux kernel 5.19+ for guest and host support (earlier for partial upstream, full for 6.x); requires TDX-aware VMM like QEMU 7.x+; Windows Server 2022 (specific Azure SKU) for guest use	Linux kernel 5.10+ for SEV, 5.11+ for SEV-ES, 5.14+ for SEV-SNP; requires KVM/QEMU with SEV-aware patches; Windows Server 2019+ (SEV) or Azure Confidential VM images
Virtual Machine Monitor (VMM)	QEMU 7.x+ with KVM; Intel TDX guest and host patches; libvirt support	QEMU 6.2+ (SEV-SNP in 7.0+ preferred); libvirt and KVM SEV patches
Cloud Provider Support	Azure Confidential VMs (preview in 2025), Google Cloud Confidential VMs roadmap	Azure Confidential VMs (production), Google Cloud, Oracle Cloud, IBM Cloud
Hypervisor Support	KVM (upstream), VMware (planned/experimental), Hyper-V (Azure only)	KVM (upstream), VMware (with support for SEV-ES), Hyper-V (Azure only)

TABLE 5. ENUMERATION OF REQUIREMENTS TO EXPLOIT TDX AND SEV-SNP TEE

Conclusions

- Our SECaaS hardening can leverage SGX as an optional flavor to secure our Systemic routine, using a hybrid layout motivated for a seamless workload development workflow.
- Our SECaaS hardening can leverage TDX and SEV-SNP with a lesser relevance in terms of usefulness and sustainability. The motivation shall only emerge for specific and highly security-sensitive use cases with an insider threat model, plausible with large TCB content.

4. Robust, Sustainable and Privacy Preserving AI in DESIRE6G

Implementation details of AI/ML components developed across different layers of the architecture (see Figure 1 and Figure 2) are discussed in this section. More specifically, details of AI/ML components within Intent Based Orchestration layer, Optimization and Network Control layer, and Physical Infrastructure layer are documented. It is worth highlighting that the AI/ML components in Intent Based Orchestration layer and Optimization and Network Control layer are related to non RT (section 4.1) and nRT decision making (section 4.2), respectively. In addition, the AI/ML components in Physical Infrastructure consist of algorithms that leverage fast physical layer operations, such as decisions pertaining to resource management, among others (section 4.3).

4.1 Non-RT Decision Making at SMO

In this section, we explain the Robust, Sustainable and Privacy Preserving AI/ML methods used for non-RT decision making at the SMO. See [38] [39] [40] for original articles.

4.1.1 ML-Based Assisted SLA Decomposition

A network slice can span across various domains of the network (such as the access, core, and transport networks) and may be implemented across multiple operators or infrastructure providers. As a preliminary step toward service instantiation, the E2E SLA must be broken down into specific SLOs by the OE for each of these network/administrative domains (hereafter referred to as domains) [38] [39]. These individual SLOs then facilitate resource allocation. AI-driven SLA decomposition is anticipated to be central to automating complex 6G business processes [41].

The OE system determines the SLA breakdown for each incoming service request, albeit with only limited insight into the current state of the infrastructure within each domain at the time of decomposition. Instead, the OE relies on historical service admission control data from each domain, provided by local IML or software-defined networking (SDN) controllers, based on prior service requests. As exemplified in Figure 30 in the context of DESIRE6G for the RAN/transport/core network, with an SLA involving the maximum delay for the E2E service. We further assume that SLAs with more stringent

SLOs (e.g., lower latency, higher throughput) are less likely to be accepted, creating a partial ordering for SLAs and a monotonic behavior of service acceptance [42].

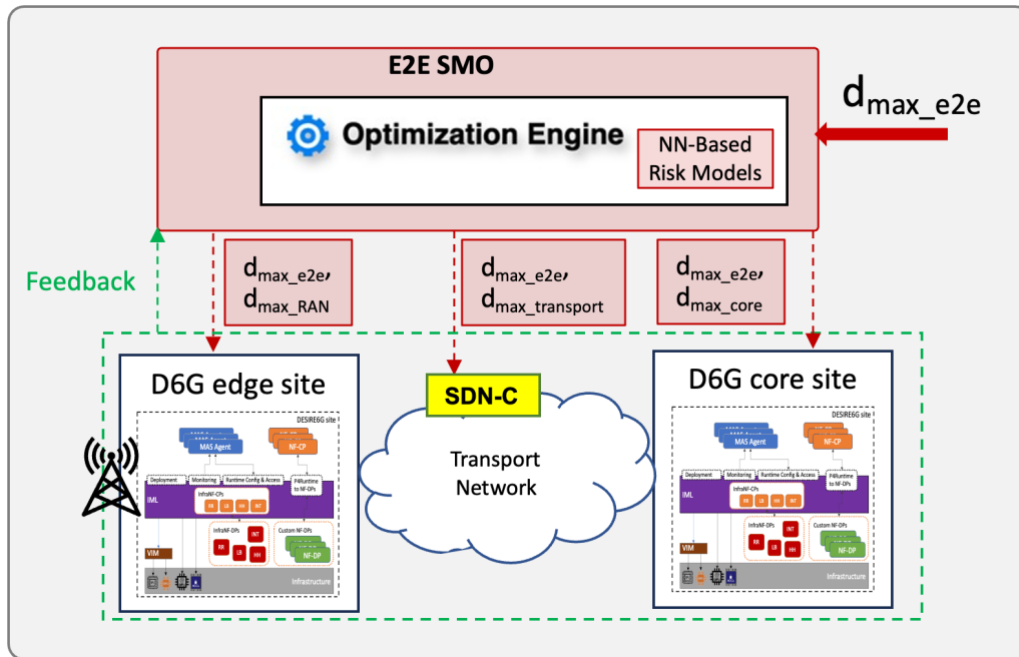


FIGURE 30 SLA DECOMPOSITION.

Several SLA decomposition methods employing heuristics have been studied in [43]. Authors in [44] present an E2E SLA decomposition system that applies supervised machine learning to break down E2E SLAs into access, transport, and core SLOs. In our previous work [42] [38], we tackled the problem with a two-step approach, which is a combination of machine learning and optimization-based solutions.

While the approaches in [42] and [38] have demonstrated success, they do not consider the inherent dynamicity of the system, which is critical for precise SLA management across network domains. This dynamicity is shaped by several factors, including variations in traffic intensity, shifts in user behavior, and fluctuating network conditions. As these factors evolve, the dynamics within each domain also change, potentially affecting the decision-making process of domain controllers. To address this gap, we propose an online learning-decomposition framework [39], specifically tailored for SLA management in dynamic, multi-domain environments. This is especially well-suited for the dynamic DESIRE6G environment.

Particularly, we propose an online learning-decomposition framework termed Real-time Adaptive DEcomposition (RADE), which is capable of running stable updates of the model as well as providing resilience against noisy samples (see Figure 31). The following are the key components of RADE:

- **Base model:** Following the design in [38], the base model is an NN. To account for monotonicity, we employ Absolute Weight Transformation (AWET) approach that shows prominent performance. AWET ensures that the weights remain non-negative, a sufficient condition for an NN to be monotonic, while allowing the model's weights to be optimized freely during training.
- **Online update:** Unlike traditional static models, which are trained once and applied indefinitely, our approach involves periodic updates to the model based on the most recent feedback collected within each discrete time step. As illustrated in Figure 31, the loop begins with a base model and employs simple Online Gradient Descent (OGD) to perform updates. The continuously updated model is then used for real-time decomposition, ensuring that the system adapts promptly to the latest conditions.
- **FIFO memory buffer:** Updating the model solely based on the most recent observations can lead to instability. For instance, the model may overfit when the feedback data is sparse, or learning may be compromised if feedback data contains errors. To mitigate these issues, we propose using a FIFO buffer with finite capacity for storing feedback. The FIFO buffer ensures a more stable and reliable learning process by maintaining a portion of historical feedback alongside all recent feedback.

Note that the decomposition step follows the work in [42] and [38], where the sequential least squares programming (SLSQP) algorithm optimizer is applied.

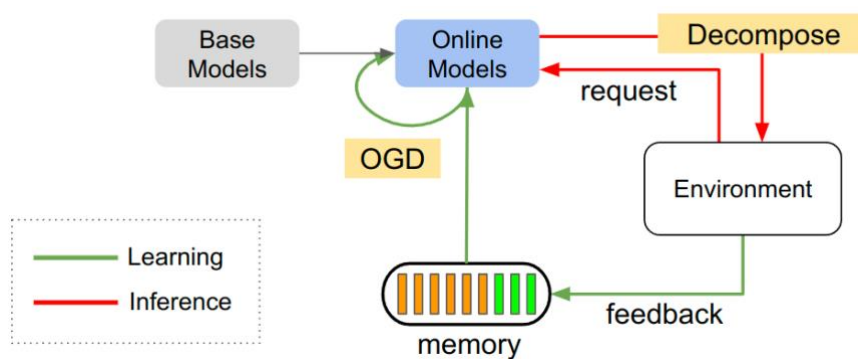


FIGURE 31 ILLUSTRATION OF RADE FRAMEWORK.

Figure presents the average E2E acceptance probability across different arrival rates for four methods: Static, RADE*, RADE, and OPT. The arrival rates vary between 0.3, 0.5, and 0.7 requests per time unit, representing different traffic intensities in the system. The Static method involves a one-time training of risk models, whereas RADE* is a variant of RADE that omits the FIFO memory buffer, and OPT

represents the optimal theoretical performance, serving as a benchmark for comparison. The RADE method, which includes the FIFO memory buffer for maintaining historical feedback, outperforms both Static and RADE* across various arrival rates, highlighting the importance of the use of the FIFO buffer to enhance the robustness and stability of the framework, particularly in scenarios with limited feedback.

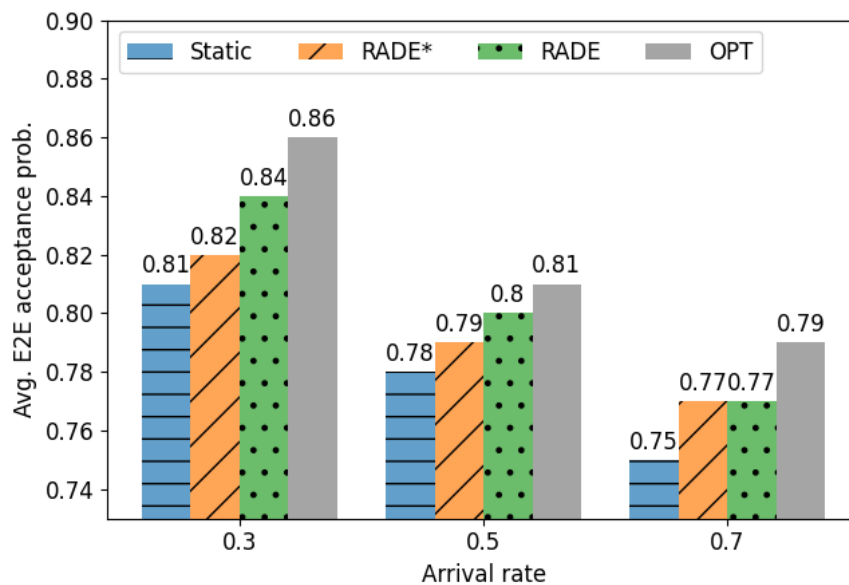


FIGURE 32 AVERAGE E2E ACCEPTANCE PROBABILITY VS. ARRIVAL RATE.

Figure 33 illustrates the average E2E acceptance probability for the RADE* and RADE methods under varying corruption rates (0.1, 0.2, and 0.3), where each feedback is corrupted (i.e., the request is always rejected). As the rate increases, the performance of RADE* significantly deteriorates, while RADE consistently maintains a higher acceptance probability across all corruption rates. This figure clearly demonstrates again that RADE is more resilient to corrupted feedback compared to RADE*, due to the use of the FIFO memory buffer.

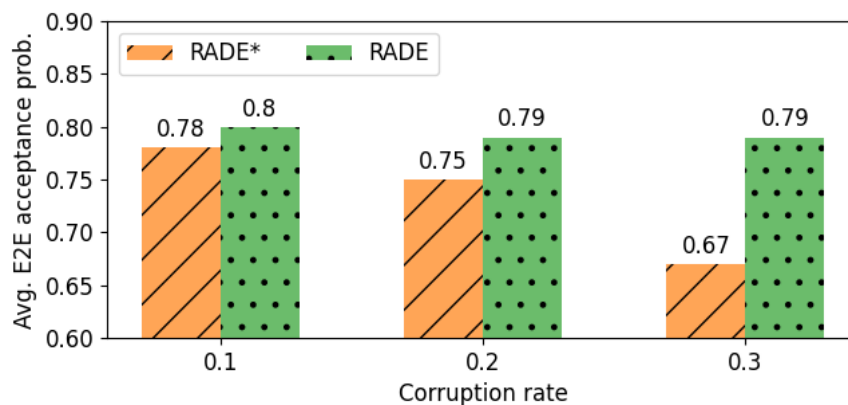


FIGURE 33 AVERAGE E2E ACCEPTANCE PROBABILITY VS. CORRUPTION RATE AT ARRIVAL RATE=0.5.

4.1.2 Transformer-Empowered Actor-Critic Service Function Chain Partitioning

In 3GPP networks, logical segmentation enables per-service customization, supported by SDN for programmable control, NFV for elastic deployment of network functions, and SFC for chaining functions into flexible service graphs [45]. Orchestration platforms integrate these technologies to automate lifecycle management, reducing CAPEX/OPEX through efficient resource use and simplified operations. DESIRE6G extends these principles across heterogeneous domains, enabling E2E service lifecycle management. However, the integration of different technology and administrative domains, dynamic service requirements, and real-time decision-making introduces additional orchestration complexity. The Service Function Chain Partitioning (SFCP) problem arises in this context when an SFC must be deployed across different domains. Instead of orchestrating the entire chain end-to-end, the SFC graph is divided into sub-graphs that can be independently managed within each domain. The key challenge is to determine an optimal partitioning strategy that satisfies operational objectives and constraints while reducing orchestration complexity, enabling localized decision-making, and supporting multi-domain deployment without requiring full visibility of each domain's infrastructure.

In DESIRE6G, the requested **SFC**, represented as an annotated service graph as described in Section 2.2, is partitioned at the SMO into sub-graphs i.e., SFC segments. To address this SFCP problem, we introduce the Sequence-aware Differentiable Actor-Critic RL (SDAC), a novel method that integrates Transformer layers within a differentiable actor-critic RL framework. The objective is to derive a strategy that maps VNFs and their virtual links onto substrate network domains while ensuring that: (i) (aggregated) resource constraints are satisfied, (ii) end-to-end latency requirements are met, and (iii) the long-term acceptance rate of service requests is maximized.

The focus of this work lies solely on partitioning of SFCs at the SMO level, employing the OE module. The process involves the interaction among the components of the SMO (see Section 2.2). The workflow (Step 3 in the AI/ML Workflow, Section 2.2.1) proceeds as follows: after receiving an SFC request, the system extracts resource demands and QoS requirements (latency, bandwidth). It then retrieves the aggregated state of each DESIRE6G site and their interconnections, combining this with the request to form a joint state. This state is processed by the OE's decision-making module, which determines the mapping of VNFs across sites. The resulting sub-graphs (exported as annotated YAML files) are distributed by the SO, and each site instantiates its assigned VNFs at the IML/K8S level, ensuring the SFC is deployed according to the partitioning decisions.

Given the NP-hard nature of SFCP [46], we reformulate the problem as a Markov Decision Process (MDP), enabling us to leverage RL techniques for a scalable and efficient solution. Our proposed DRL framework, SDAC, incorporates Transformer encoders in both the actor and critic networks, leveraging their ability to model interdependencies among VNFs and make informed, coordinated decisions, as shown in Figure . Details on the MDP formulation and the stages of the SDAC pipeline are available in [47]

Earlier work on service SFCP relied on optimization-based methods such as ILP, MIP (e.g., [48] [49] etc.), often complemented by heuristics or meta-heuristics (e.g., Tabu search [50], ILS [51]) to improve tractability. While these methods can provide near-optimal solutions, exact formulations suffer from exponential complexity, and heuristics often trade optimality for scalability. More recent data-driven approaches, including reinforcement learning (PPO-based [52], multi-agent RL [53]) and graph neural networks [54] offer improved adaptability and scalability. Despite significant advancements, these works predominantly focus on either sequential or parallel VNF placement, each with inherent limitations. Sequential placement of VNFs, while offering strong coordination, is time consuming and constrained by the interdependence of prior decisions, which can impact overall efficiency [53] [55]. Parallel VNF placement offers faster deployment, but suffers from weak coordination among VNFs, as individual placement decisions are made without full awareness of the broader context. Despite the success of Transformers in other domains, their application to real-time orchestration in networking is still limited. The proposed SDAC differs by leveraging self-attention to capture inter-VNF dependencies while enabling parallel, sequence-aware decision-making, achieving scalability without losing coordination.

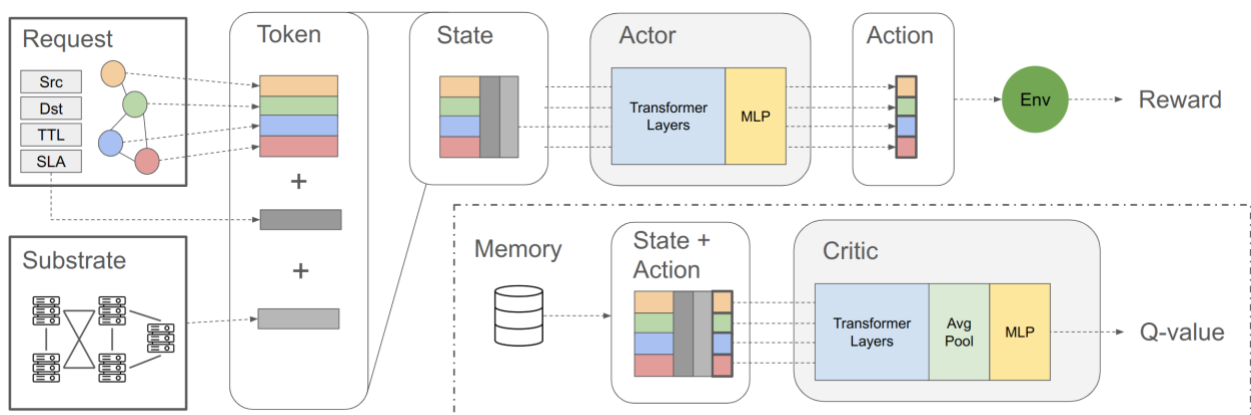


FIGURE 34. OVERVIEW OF THE SDAC FRAMEWORK WITH TRANSFORMER-BASED ARCHITECTURE FOR SEQUENCE-AWARE SFC PARTITIONING.

We evaluate the proposed framework in [30] against several baselines: **Greedy Policy (GP)**, which assigns the SFC to the DESIRE6G site with the highest remaining capacity; **Iterated Local Search (ILS)**, a meta-heuristic optimization method that iteratively refines solutions by applying local search techniques with perturbation to escape local optima; **Risk-Aware ILS (RAILS)**, which incorporates online risk models to predict acceptance probabilities; **Sequential DDQN (seqDDQN)**, mapping VNFs sequentially Double Deep Q-Networks (DDQN); and **Parallel DDQN (paraDDQN)**, a DDQN-based policy which maps all VNFs simultaneously to accelerate decision-making.

A total of 10,000 requests is considered. The request size and resource demand are sampled from a uniform distribution. Arrivals follow a Poisson process with mean rate $\lambda=0.05$, where λ varies over time according to a sinusoidal function. Service lifetimes are sampled from an exponential distribution, while SLA latency requirements are drawn from a uniform distribution. Details on the simulation environment and scenarios are detailed in [47].

Figure 35 presents the acceptance rate of the different algorithms over time (top) and the corresponding arrival rate of SFC requests (bottom). Acceptance rate is a critical performance metric as it measures the proportion of SFC requests that arrive at the SMO and are admitted to the DESIRE6G infrastructure over the entire evaluation period. In the early phase of the simulation, the acceptance rate is relatively low for all algorithms as the arrival rate of SFC request reaches its peak during this period. As the arrival rate decreases, acceptance rates steadily improve, leading to an improvement in acceptance rates. Among the algorithms, the proposed SDAC consistently outperforms the others throughout the simulation.

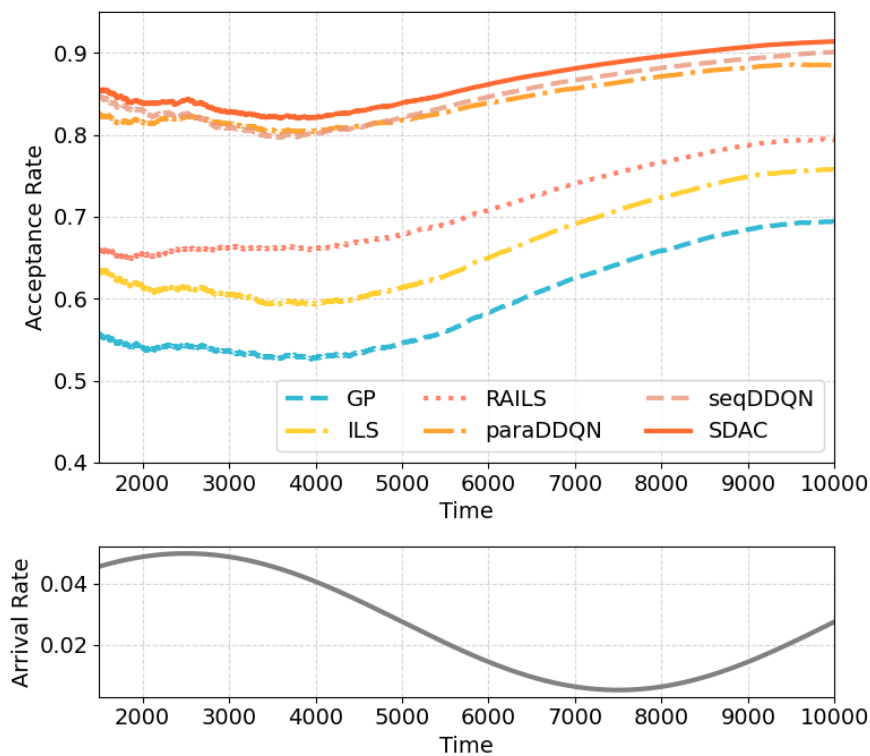


FIGURE 35. ACCEPTANCE AND ARRIVAL RATE OVER TIME.

Indicatively, the proposed SDAC consistently achieves the highest acceptance rate throughout the simulation. seqDDQN shows competitive performance but remains below SDAC, with paraDDQN slightly behind. RAILS performs poorly in the early phase due to the risk models that require time to converge, but later surpasses ILS, although it remains weaker than the DRL-based methods. The GP yields the lowest acceptance rate, highlighting its inefficiency in resource management. These results confirm the effectiveness of SDAC in sustaining high acceptance rates under varying request load.

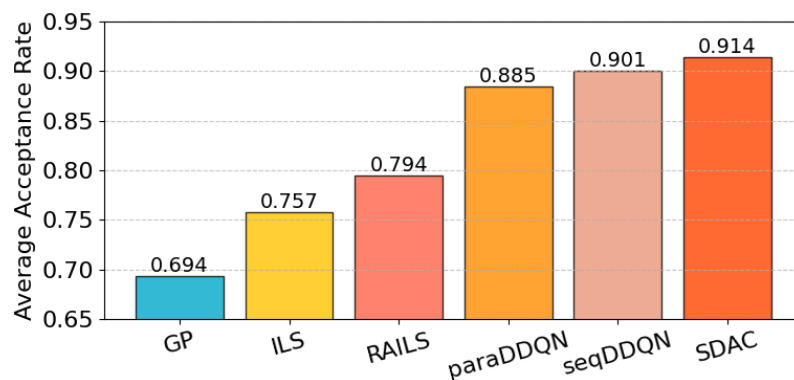


FIGURE 36. AVERAGE ACCEPTANCE RATE.

In continuation, Figure 36 shows the average acceptance rate achieved by the different algorithms. SDAC exhibits the highest average acceptance rate, while both seqDDQN and paraDDQN achieve competitive acceptance rates, although lower than SDAC. The stronger coordination of decisions in seqDDQN results in a performance advantage over paraDDQN. RAILS and ILS fall into the second tier, consistent with the results in the previous figure. GP again achieves the lowest acceptance rate.

Additional results including how computational resources are utilized across different DESIRE6G sites, rejection causes for service requests, inference time etc. are available in [29].

4.1.3 Algorithmic Developments within OE

This section expands on the algorithmic developments within the proposed OE framework, described in Section 2.2. The developments can be separated into two distinct sections: Optimization models and model selection.

4.1.3.1 Model Pool Optimization Models

The Optimization models are algorithms that they optimize a service, based on a specific KPI and with a specific method. We have developed these models based on the current state of the art as partitioning baselines.

Optimization Model (Partitioning) Baselines

- **Single DDQN RL Agent:** An RL agent with global network visibility that learns an optimal partitioning policy, by minimizing overestimation bias in Q-value updates [56].
- **Multi Agent DDQN RL:** An extension of DDQN where multiple RL agents operate in parallel per domain, each responsible for local operations. The agents act cooperative or competitive and learning to improve network-wide service partitioning [55], [53].
- **Greedy Partitioning:** A baseline heuristic approach that iteratively selects the locally optimal partitioning choice step by step, without considering long-term consequences, simulating models that aim for simplicity and low computational cost.
- **SLA Aware Partitioning:** A heuristic baseline that is partitioning services by optimizing contractual constraints first.
- **Logic Based Partitioning:** A rule-driven baseline strategy where partitioning decisions follow predefined logical conditions, simulating simple approaches.

- **Resource Based (CPU) Partitioning:** Resource-centric greedy method where partitioning is determined primarily by CPU utilization and availability.
- **Random Partitioning:** Non-deterministic baseline where partitions are decided randomly. Used to evaluate the relative advantage of more sophisticated algorithms.

4.1.3.2 Model Selection

The OE serves as a generic optimization framework at the SMO layer that assesses the incoming service request with its respective SLA and objectives and picks the best approach from the Model Pool, as discussed in Section 2.2.

Algorithm selection was first defined by Rice in [57] as a problem of selecting the most appropriate algorithm from a set of available algorithms, for a given problem instance. Related work can be divided into traditional and recent approaches. Traditional methods often relied on scoring and ranking techniques to sort and cluster algorithms [58] [59]. For example, Kadioglu et al. [58] applied similarity between problem and algorithm instance features. Recent works, driven by the advances in AI and, more recently in LLMs, have explored the use of these technologies for algorithm selection. A notable example, Wu et al. [60], proposed an LLM-based algorithm selection framework that extracts feature representations from both algorithms and problem instances to identify the most suitable algorithm. They demonstrated with results that including algorithm representations significantly improves the selection accuracy compared to traditional methods that rely only on problem features, particularly when the training data is limited.

The model selection of the OE is based on LLM-generated model representations of Python code and Abstract Syntax Tree (AST) complexity analysis [61] for all models of the Model Pool.

Specifically, the representation of each model used for selection is created in two phases:

1. **Complexity report:** The OE iterates over all models and algorithms in the pool, generating an algorithmic complexity report. The approach used is AST-based complexity analysis, which is used to estimate the computational time required, useful to meet online requests.
2. **LLM-based code representation:** An LLM processes the Python code of each model and produces a descriptive text representation.

These two outputs are combined into a unified textual description of each mode, called a Model Description. These descriptions are passed to the LLM model selector, which evaluates the options and chooses the most appropriate model for the given optimization.

The LLM prompt used for model selection is composed of four main elements: the Model Descriptions, the requested optimization objective or SLA, the abstracted network state, and an optional custom user prompt that can be added to create a selection bias toward specific models (e.g., toward energy-efficient models).

Several methods were used for the LLM-based selection of the model to evaluate this work. Specifically:

- **GPT-5 (Online):** Semantic precision and strong reasoning for ambiguous SLA objectives analysis. It has higher latency and privacy concerns due to API calls.
- **GPT-4o (Online):** Balances accuracy and speed with slightly reduced reasoning depth compared to GPT-5.
- **GPT-4o-mini (Online):** Offers low latency and cost. It sacrifices some semantic nuance compared to GPT-4o.
- **Llama3.2:1b (Local):** Lightweight open-source local model that preserves privacy and runs with minimal hardware requirements. It presents lower reasoning depth than the other selection models.
- **Llama3.2:3b (Local):** Medium-sized local model with better capabilities than the 1b variant. Requires more computing resources.

Indicatively, we evaluate the proposed model selection approach on a service graph partitioning use case over 3 DESIRE6G sites. Each service request includes SLA constraints (e.g., latency, throughput) and is processed using the LLM-based model selector. Four candidate selectors are considered: GPT-4o, GPT-4o-mini, Llama3.2-1b, and Llama3.2-3b. Each experiment runs for 100 service requests. The evaluation considers two key metrics: the service graph partitioning success rate, reflecting the percentage of service requests deployed without SLA violations, and the corresponding inference time of the model selection process.

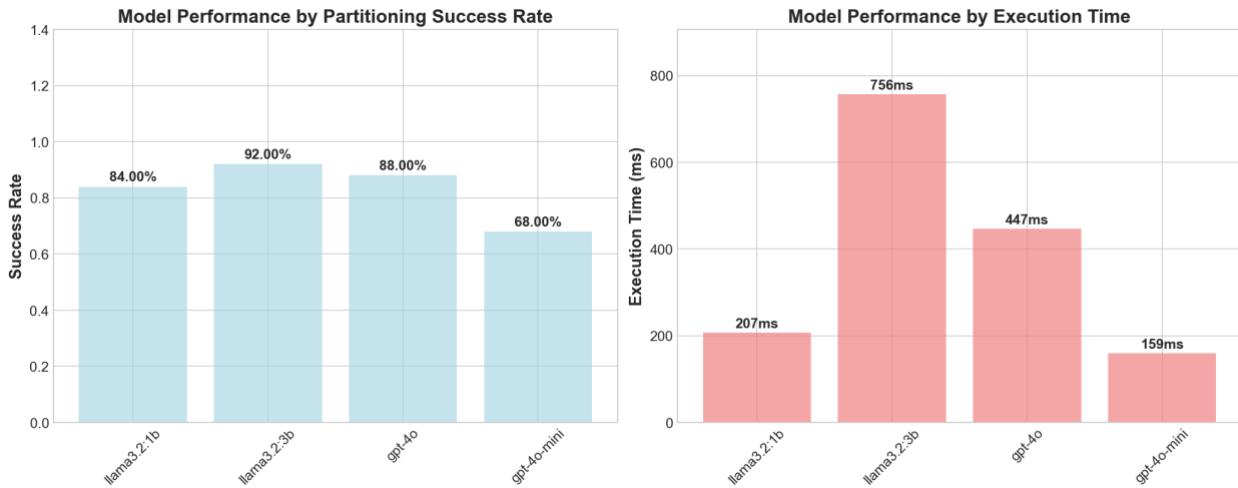


FIGURE 37: PARTITIONING SUCCESS RATE AND AVERAGE INFERENCE TIME PER MODEL.

Partitioning success rate and inference time varied across LLM models, while there is an obvious trade-off between success rate and inference time, that directly reflects the characteristics of the evaluated LLMs. The lightweight local model (Llama3.2-1b) achieves comparable performance to the other models while maintaining a short inference time. The model Llama3.2-3b provides the highest success rate, benefiting from stronger reasoning, at the cost of significantly higher inference time (4x). GPT-4o achieves near-top accuracy with lower latency than Llama3.2-3b, leveraging optimizations in its cloud deployment while GPT-4o-mini delivers the fastest inference time with the lowest success rate. Overall, these findings indicate that the OE can be tuned to the most appropriate selector according to the requirements of the operator (i.e., the best trade-off between response time and accuracy). The integration of AST-based complexity analysis with LLM-driven model selection is an ongoing effort, with extended evaluations currently in progress beyond the indicative results presented in this report.

4.2 Near-RT Decision-Making at MAS

In this section, we first detail the procedures for the flow provisioning phase, formally describing the DRL engine behind autonomous flow routing and the subsequent nRT flow operation [30]. Then we present the nRT elastic scaling methods [62] [63].

4.2.1 Autonomous Flow Routing and Subsequent nRT flow operation

Table 6 summarizes the notation used in the rest of the section, which is originally from [30].

TABLE 6 NOTATION (MAS CONFIGURATION ALGORITHMS).

Notation	Description
$x(t)$	Input traffic at time t .
$d(t)$	E2E delay measured at time t .
$s(t)$	State at time t .
P	Set of available routes for the flow.
k_p	Capacity of route p .
c_p	Cost of route p .
$y_p(t)$	Traffic routed through route p at time t .
$a_p(t)$	Fraction of traffic to be routed through route p at time t .
$r(t)$	Reward at time t .
$r_{\text{delay}}(t)$	Reward component for the obtained delay at time t .
$r_{\text{cost}}(t)$	Reward component for the routing cost at time t .
$\alpha_{\text{delay}}, \alpha_{\text{cost}}$	Weights for the rewards in the multi-objective reward function.
β	Fixed penalty for violating the maximum delay.
d_{max}	Maximum delay to be ensured for the flow.
x_{max}	Maximum traffic flow.
p_{max}	Maximum number of routes for the flow.
$G(V,E)$	Graph with the current network state, where V is the set of nodes and E the set of links connecting two nodes.

4.2.1.1 MAS Configuration

Let us start with the very first procedure that is executed at flow provisioning time Figure 38. Upon the reception of a new flow provisioning request (*req*), the SMO runs Algorithm 1 given in Table 7 to compute the set of allowable routes P . Algorithm 1 receives: *i*) the current network state, summarized in graph $G(V,E)$, where V is the set of nodes and E the set of links, and *ii*) the request including source s and target t nodes, maximum traffic (x_{max}), delay to be guaranteed (d_{max}), and maximum number of allowed routes p_{max} . The algorithm first discards those links with residual capacity below x_{max} (lines 1-3 of Algorithm

1). Next, the k -shortest path algorithm is used to compute k distinct routes on the resulting graph G_{aux} (lines 4-8). The minimum expected delay d_{min} (considering both *transmission* and minimum *queuing* and *processing* delay) is computed and used to discard those routes that cannot meet d_{max} . Finally, the remaining routes are sorted by multiple criteria. In this way, the returned set of routes includes those with expected high QoS, while providing high diversity, so agents can choose among alternative options for nRT decision making. After route computation, relevant parameters to each $p \in P$, such as the routing cost and capacity are added, which are necessary for autonomous flow routing operation. Note that the complexity of Algorithm 1 is dominated by that of the k -Shortest Path (SP) algorithm.

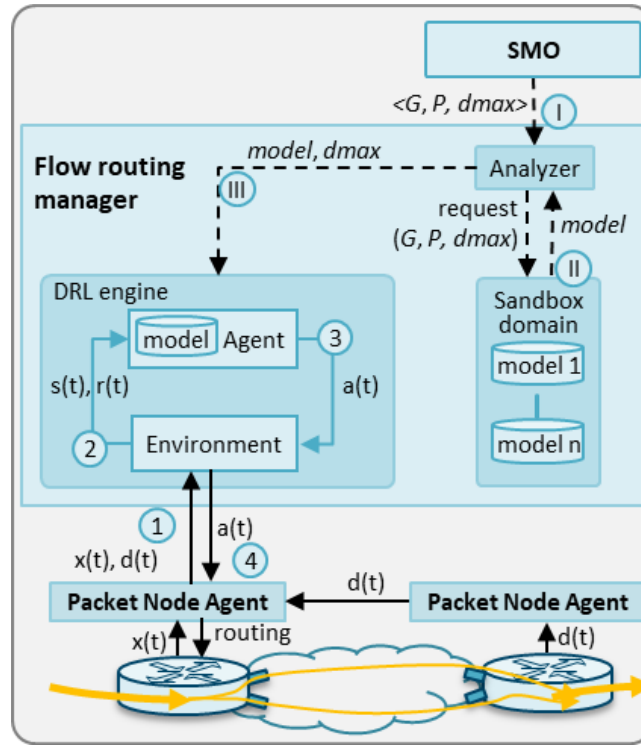


FIGURE 38 FLOW OPERATION UNDER TRAFFIC UNCERTAINTY.

Next, the MLFO in the SMO initializes the flow routing agents, which contain the following modules (see Figure 38): *i*) the *DRL engine* as described below; *ii*) the *sandbox domain*, that contains pre-trained models and provides support for offline learning; and *iii*) the *analyzer*, in charge of evaluating the performance of models in operation and triggering model updating actions in case of poor performance. Once the flow manager is instantiated, the workflow (see Roman numerals) in Figure is executed. The topology G , the set of allowable routes P computed by Algorithm 1, and QoS requirement d_{max} are provided to the analyzer (see I in Figure 38). Note that G also contains the maximum traffic volume for

the background traffic that is expected for each packet link. Then, the analyzer requests to the sandbox domain an initial model (see II). Without loss of generality, we assume that the initial model has been pretrained offline, reproducing a network with the same topology G and routes P as those requested. A generic input traffic, e.g., a sinusoidal daily pattern with some random variation, as well as constant background traffic according to configured maxima in G are also assumed. Note that to enable offline training, the specific traffic characteristics are not well known at that time. Therefore, the selection of the generic traffic must verify that pretrained models provide the committed performance when they enter into operation. Finally, the sandbox domain returns the initial model that will be loaded in the DRL engine before operation (see III).

TABLE 7 ALGORITHM 1. ROUTE COMPUTATION ALGORITHM.

INPUT: $G(V,E)$, req= $\langle s, t, x_{max}, d_{max}, p_{max} \rangle$	
OUTPUT: P	
1:	$G_{aux}(V_{aux}, E_{aux}) \leftarrow G$
2:	for each $e \in E_{aux}$ do
3:	if $e.cap < x_{max}$ then $E_{aux}.pop(e)$
4:	$P \leftarrow k\text{-SP}(G_{aux}, s, t)$
5:	for each $p \in P$ do
6:	$p.dmin \leftarrow \text{transmDelay}(p) + qmin$
7:	if $p.dmin > d_{max}$ then $P.pop(p)$
8:	$\text{sort}(P, \langle dmin, similarity \rangle, \text{ASC})$
9:	return $P[1 .. p_{max}]$

4.2.1.2 Autonomous Flow Routing, Running in the Agents

Among different DRL techniques, we selected Twin Delayed Deep Deterministic Policy Gradients (TD3). TD3 is an off-policy DRL algorithm that uses a pair of *critic* DNN and an *actor* DNN that is updated with some delay. Hereafter, we refer to the set of critic and actor DNNs that run inside the DRL engine as the *model*. The model is in charge of dynamically and autonomously deciding which fraction of traffic is routed through each of the allowable routes P that can support the flow.

During nRT operation, flow routing agents compute the state, the reward, and the actions periodically (e.g., 1 sec.). Recall that $x(t)$ is the input traffic and $d(t)$ is the E2E delay, measured at time t (see Table 6).

State $s(t) > 0$ is defined as the input traffic scaled by the average capacity of the available routes P , where (k_p) is the capacity of route $p \in P$. In particular, we have

$$s(t) = x(t) \cdot \frac{\sum_{p \in P} k_p}{|P|}.$$

Each action $a(t) \in [0,1]^{|P|}$ is a $|P|$ -dimensional vector, where every component specifies the fraction of input traffic to be routed through route p . Then, $y_p(t)$, the traffic routed through p is given by

$$y_p(t) = a_p(t) \cdot x(t).$$

The reward function $r(t)$ should penalize those actions that resulted into poor QoS or increased network cost. In consequence, a multi-objective reward function with two reward components has been considered to account for the obtained delay (r_{delay}) and for the routing cost (r_{cost}), being α_{delay} and α_{cost} the weight of each component. More specifically,

$$r(t) = \alpha_{delay} \cdot r_{delay}(t) + \alpha_{cost} \cdot r_{cost}(t),$$

where,

$$r_{delay}(t) = \begin{cases} -\beta - \frac{d(t)}{dmax}, & d(t) > dmax \\ 0, & d(t) \leq dmax, \end{cases}$$

and

$$r_{cost}(t) = -x(t) \cdot \sum_{p \in P} a_p(t) \cdot (c_p/k_p).$$

Considering that $dmax$ needs to be ensured for the flow, the reward related to the obtained delay can be defined as follows, where β is a fixed penalty for violating the maximum delay. Finally, the reward component for the routing cost is related to the proportion of traffic sent through each of the routes, as well as to the ratio of cost (c_p) and capacity.

4.2.1.3 Service Operation

Figure details also the workflow for autonomous flow routing with delay requirements in a single domain scenario (see Arabic numerals). At every time interval t , input traffic $x(t)$ and delay $d(t)$ are collected from the source packet node agent and fed into the DRL environment (1). Note that $x(t)$ can be directly measured at the source packet node, whereas $d(t)$ is computed at the destination and sent to the source node agent. As previously introduced, the environment block is in charge of computing the current state $s(t)$ and reward $r(t)$ and sending them to the DRL agent (2), which is in charge of both

learning and decision-making (3). Action vector $a(t)$, with the fraction of traffic to be routed through each of the routes in P , is forwarded to the packet node agent (4). This agent is responsible for translating such proportions into suitable packet node routing configuration and of configuring the routing tables.

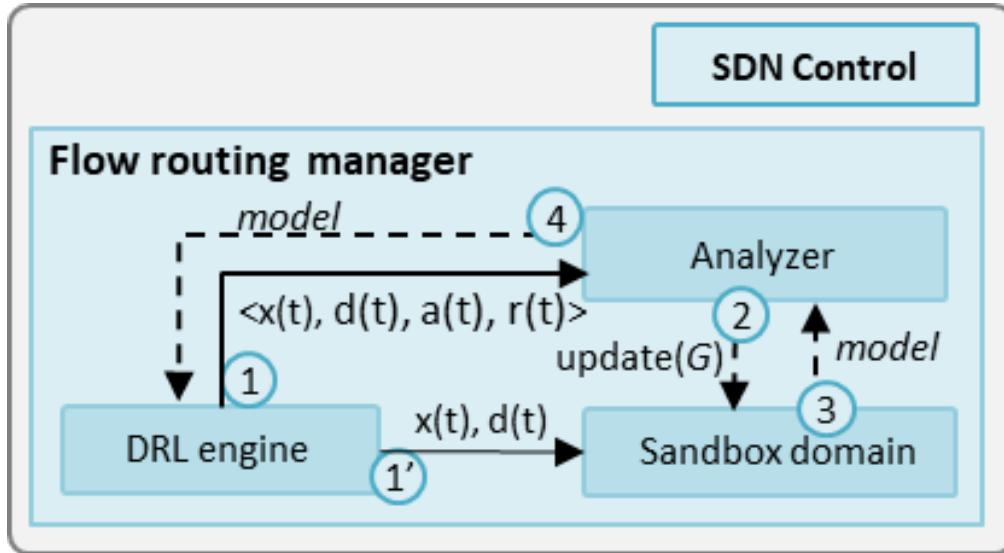


FIGURE 39 FLOW OPERATION UNDER TRAFFIC UNCERTAINTY.

The pre-trained model that is loaded in the DRL engine in the provisioning phase is smoothly improved online from the decisions and actions performed during operation. However, the uncertainty in the input and, more important, the background traffic can lead to poor performance. For instance, if the maximum background traffic is significantly underestimated, large d_{max} violation can occur, which cannot be corrected by online learning. Then, to overcome such issue, Figure details the proposed workflow that runs in parallel to DRL operation and is intended to improve autonomous operation under traffic uncertainty.

Let us consider that every time the DRL takes and evaluates an action, it pushes tuple $\langle x(t), d(t), a(t), r(t) \rangle$ to the analyzer (labeled 1 in Figure). At the same time, the monitoring data $x(t)$ and $d(t)$ are also fed to the sandbox domain to tune the offline platform according to real traffic and performance measurements. Then, the analyzer block evaluates the performance of the current model and, if needed, requests the sandbox domain to provide an update of the model with a different configuration for the background traffic (2). The offline trained model that fits better with the new scenario is provided (3) and fed to the DRL engine (4).

The procedure for detecting poor performance due to background traffic misconfiguration (and how to correct it) is illustrated in Figure . The figure shows three different cases of operation that can happen

from the same initially offline trained model. Let us assume that the sandbox domain, once the model is trained, can compute the delay range $[d_{low}, d_{high}]$, for which 95%-percentile of delay (d_{95}) is expected to oscillate. Without loss of generality, we consider that d_{95} is computed every hour with the last 60 delay values measured each minute. Note that the delay range is computed by simulation as a result of autonomous routing actions and considering the aforementioned traffic assumptions, i.e., sinusoidal input traffic and constant maximum background traffic. The behavior of d_{95} in the sandbox is illustrated in the left part of every figure (labeled *offline*), which stays within delay range, as expected.

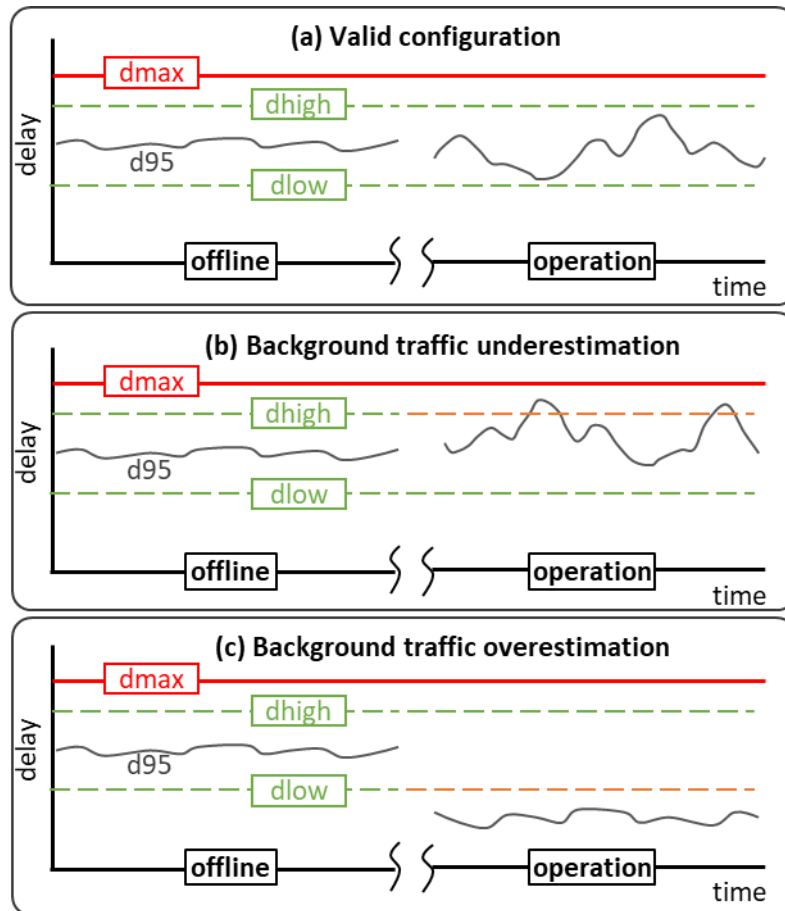


FIGURE 40 DETECTION OF BACKGROUND TRAFFIC MISCONFIGURATION.

Figure a shows a case where the DRL model has a valid performance during operation (right part of the figure). The measured d_{95} stays within the range, although it fluctuates more than that assumed during offline training (due to differences between real and simulated traffic and network state). Conversely, Figure b shows a poor performance case induced by underestimation of the real background traffic, which produces large delays, thus violating d_{high} . Note that, although in the example d_{max} is still guaranteed (not shown in the figure), it is identified as a poor performance case that will trigger model

update assuming an increase of background traffic. Alternatively, Figure c shows a case where the d_{95} stays below d_{low} , which is considered as an initial overestimation of background traffic. Here, new offline training with a reduction of maximum background traffic will be triggered.

4.2.2 Near RT Elastic Scaling Mechanisms

In DESIRE6G, effective scaling mechanisms are essential for ensuring performance, especially given the stochastic nature of traffic loads and computational resource availability [62] [63]. Elasticity, in this context, refers to the capability to autonomously adjust resource allocation as workload demands shift [64]. The objective is to maintain high service performance while optimizing the use of shared physical resources. Accordingly, we have examined the issue of scaling edge computing resources within the framework of a Cellular Vehicular-to-Network (C-V2N) service, utilizing the MAS approach in DESIRE6G.

C-V2X technology, introduced by 3GPP, enables vehicles to communicate with other vehicles (i.e., vehicle-to-vehicle (V2V)), road users (i.e., vehicle-to-pedestrian (V2P)), roadside infrastructure (i.e., vehicle-to-infrastructure (V2I)), and cloud or edge servers (i.e., vehicle-to-network (V2N)). In this work, we consider a C-V2N application supported by edge computing resources spanning multiple Points of Presence (PoPs) which are effectively DESIRE6G edge sites distributed throughout the metropolitan area of a city. The terms PoP and DESIRE6G edge site will be used interchangeably in the following.

Figure 41 illustrates the placement of vehicles' tasks to PoPs, and the number of CPUs allocated to support their respective requirements at each PoP. Both smartphone users and connected automated vehicles (CAVs) are connected to DESIRE6G edge sites. CAVs rely on V2N-based applications, such as remote driving and hazard warnings, which require V2N traffic to be forwarded and processed at the network edge to meet delay constraints. In the context of DESIRE6G, service agents may adjust edge resources dynamically (i.e., employing the DESIRE6G IML) to satisfy V2N service requirements under varying workloads. Our study addresses this elastic scaling challenge by examining *(i)* edge computing resources collocated with the base station associated with each vehicle [63], which can be found in D3.1 [1], and *(ii)* task offloading/placement across multiple edge servers distributed throughout a city's metropolitan area [62]. These two cases are interdependent. On one hand the placement of application tasks determines the computing requirements per PoP, which drive scaling decisions. On the other hand, scaling decisions define the maximum available computing resources per PoP that is further used as input for deciding on task placement.

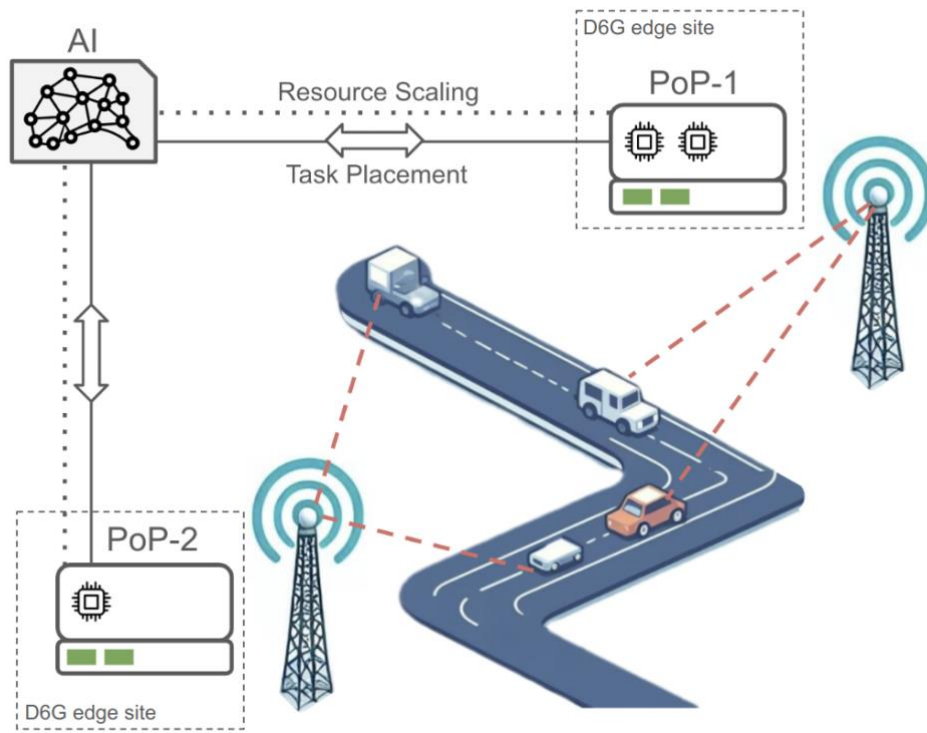


FIGURE 41 C-V2X SYSTEM OVERVIEW.

To this end, we first formulate the task placement and resource scaling decisions as a joint optimization problem, considering the latency constraints of the C-V2N application as well as cost-efficiency, in terms of the employed computing resources. The problem is initially formulated with the assumption of perfect knowledge of all future vehicle arrivals. To account for the stochastic nature of traffic loads and the availability of physical resources, we formulate the joint problem as a Markov Decision Process (MDP), assuming no prior information about future vehicle arrivals. We introduce a new DRL approach called DHPG, to support a hybrid action space that encompasses both discrete (which PoP to offload tasks to) and continuous (CPU resource allocation of DESIRE6G edge sites) actions, accommodating the different placement and scaling decisions.

Joint task offloading/placement and resource scaling problem has been extensively studied in literature [65] [66] [67] [68] [69]. Existing research can be broadly classified into optimization-based approaches (including heuristics and approximation), as well as machine learning-based approaches. To manage the complexity of the joint solution space, these problems are often tackled in a decoupled manner or considered as static scenarios, which may, however, limit the solution space. In contrast, DHPG captures the interdependencies between these tasks within a unified framework and optimizes the long-term system performance.

Specifically, as shown in Figure 42, DHPG involves (i) a state encoder that maps the high-dimensional joint state into a compact latent space, providing a noise-reduced representation with rich information, (ii) this joint representation is shared across specialized action heads, allowing each of them to make holistic and well-informed decisions independently, and (iii) we introduce the probability-as-action (PAA) approach, which allows for a differentiable representation of the hybrid action space, enabling actions from different actors to be optimized jointly with a single critic. This unified optimization results in faster convergence by ensuring more consistent gradient updates throughout the entire network.

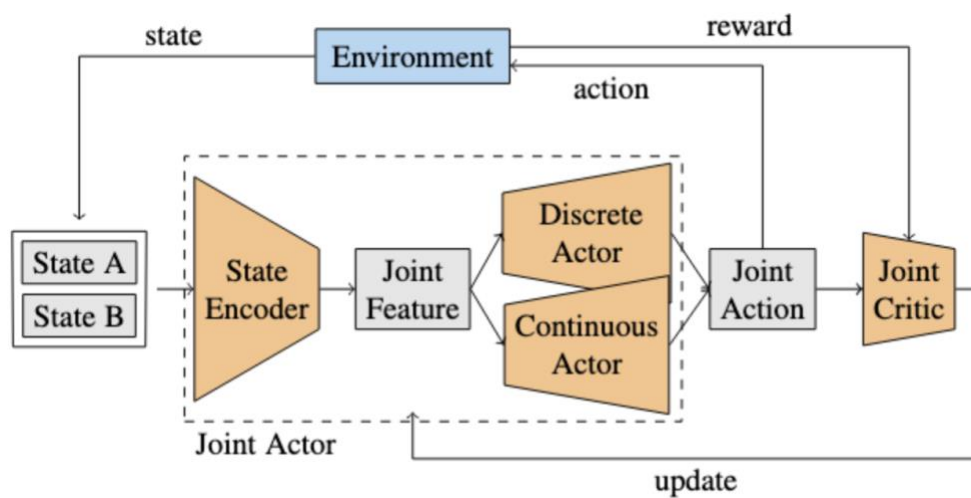


FIGURE 42 DHPG FRAMEWORK.

We compare the proposed DHPG approach to different SoA introduced in [63] [70], and [71] using simulations based on a real-world C-V2N traffic dataset. Figure shows that DHPG and DDPG obtain higher average rewards by 13% and 2% compared to the one attained by Triple Exponential Smoothing (TES), respectively while Proportional and Integral (PI) and Constant (CNST) remain below TES with a decrease in average reward of 11% and 14%, respectively. Figure illustrates that DHPG effectively allocates a minimal number of CPUs over time across varying traffic conditions while consistently meeting the target delay. Notably, the CPU allocation states exhibit a greater degree of unevenness across PoPs, highlighting the effectiveness of DHPG in deactivating unnecessary CPUs and placing tasks to PoPs with sufficient resources allocated.

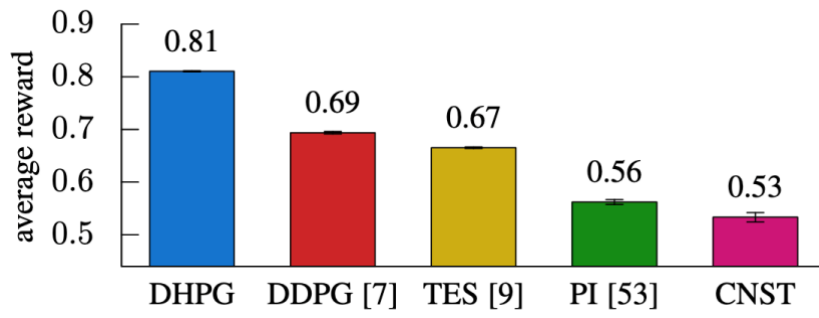


FIGURE 43: AVERAGE REWARD OVER 40 DIFFERENT TRACES OF A 5.5H INTERVAL.

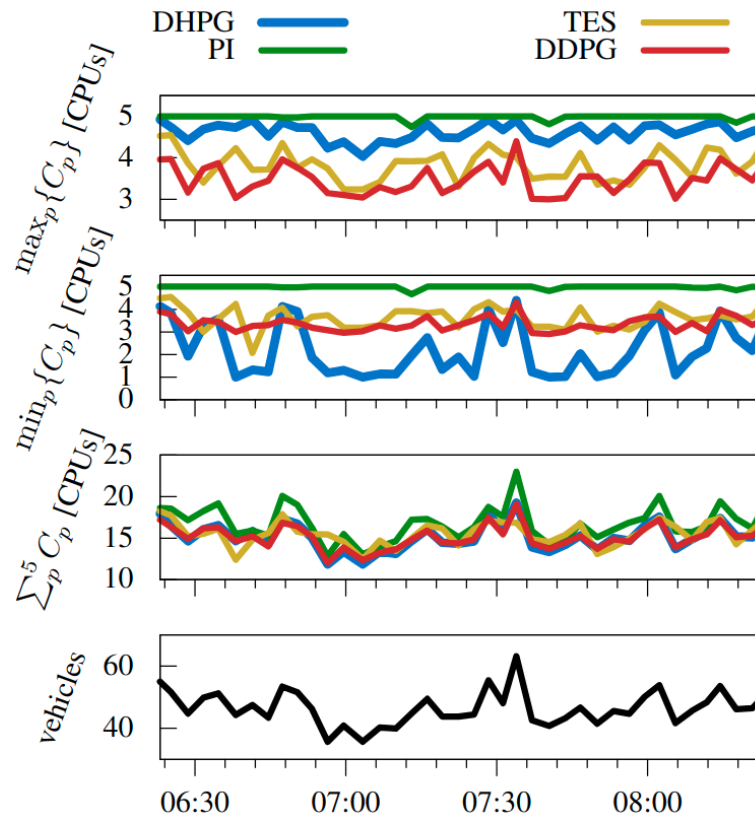


FIGURE 44 BEHAVIOUR OF ALL SOLUTIONS AS THE NUMBER OF VEHICLES CHANGES OVER TIME.

We further utilize the theoretical optimal solution on a small scale to derive optimality gaps for the selected approaches. Table 8 presents the optimality gap achieved by DHPG in a small-scale scenario. DHPG closely approaches the optimal solution, demonstrating an optimality gap of only 5.2%. In contrast, DDPG and other benchmarks exhibit optimality gaps exceeding 10%.

TABLE 8 OPTIMALITY GAP.

Algorithm	avg gap	avg #CPUs	avg load
Oracle	-	3.20	86.6%
DHPG	5.2%	3.23	90%
PI	10.4%	3.04	89%
TES	11.3%	3.04	89%
DDPG	12.1%	3.02	89.2%
CNST	77.7%	3.40	77.4%

4.3 Complementary AI/ML Algorithms

Complementary AI/ML algorithms that span over the layers of DESIRE6G architecture are discussed in the rest of this section. These algorithms are mainly related to the Physical Infrastructure Layer of the architecture, see Figure 45. It is worth noting that the implementations of some of these algorithms are presented as proof of concepts only, see D5.1 [72]. We refer the reader to references [73] [74] [75] [76] [77] [78] [79] [80], the original articles, for more details.

4.3.1 Distributed Edge Intelligence for RIS

This section first details the implementation of the relevant algorithms on the 5TONIC hardware testbed, along with the corresponding experimental results. Subsequently, we outline the fundamental message flows required for integrating these algorithms within the DESIRE6G architecture.

4.3.1.1 Implementation of algorithms

Initial developments of the algorithms that have been evaluated empirically have been documented in section 5.1 of D3.1 [1]. It is worth recalling that our proposed algorithms are designed to develop NNs to perform configuration of RISs in a wireless edge setting [73]. In addition, the architecture of NNs is designed to handle the common model training deficiencies due to dataset heterogeneity. Moreover, the AI/ML algorithms' training is conducted based on FL with intrinsic privacy preserving properties. The implementation of the related algorithms in hardware testbeds is split into 4 phases. D3.2 [2]:

- **Phase 1:** Implement the classic FL with locally deployed clients and a parameter server (PS).
- **Phase 2:** Implement the classic FL with geographically distributed clients and a PS (some clients are local, some clients are not local, and PS is not local)
- **Phase 3:** Upgrade Phase 1 to accommodate heterogeneity of the datasets.

- **Phase 4:** Upgrade Phase 2 to accommodate heterogeneity of the datasets.

The details of Phases 1 and 2 of classic FL have been documented in deliverable D3.2. In the following, we provide details of Phase 4, which focuses on addressing data heterogeneity (e.g., non IID), where FL games via causal representations within an invariant risk minimization framework are integrated [73].

Figure illustrates the integration setting of Phase 4. Note that we denote by ‘not local’ and ‘local’ the hardware equipment located at Universidad Carlos III de Madrid and University of Oulu, respectively. Moreover, all the non-local clients and the parameter server (PS) are hosted at 5TONIC.

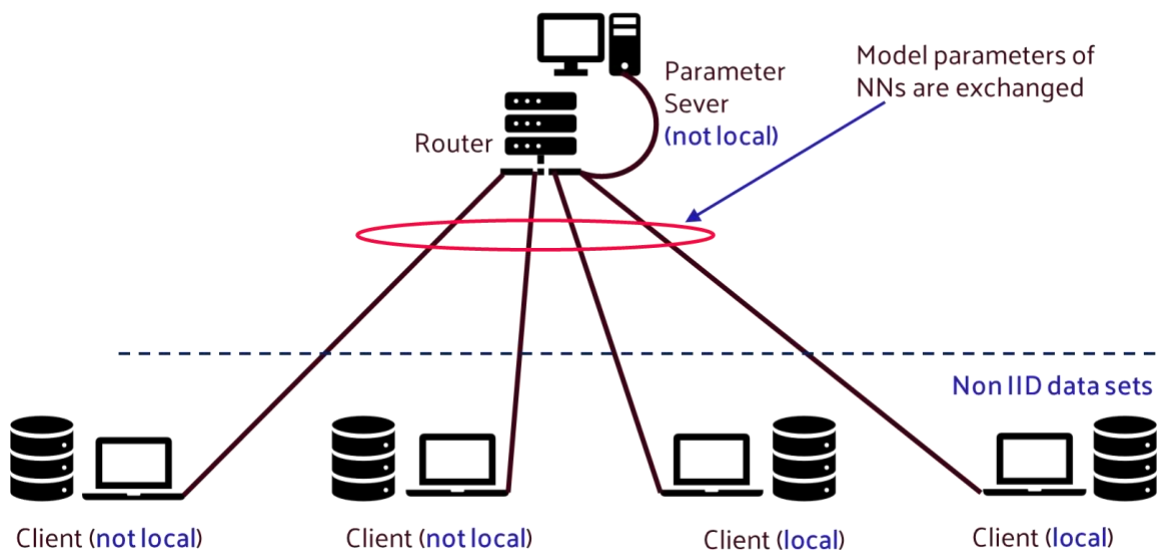


FIGURE 45 OVERVIEW OF FEDERATED LEARNING ARCHITECTURE INTEGRATION WITH 5TONIC (PHASE 4).

Upon completion of the training process, performance metrics, including test accuracy and training loss (smoothed via moving averages), are presented to the end user. Figure 46 displays such a performance plot obtained using our empirical RIS dataset, as described in Section 5.5 of D5.1 [72]. Note that a round refers to one model update performed at the parameter server. For comparison, the figure includes results from both the proposed FL games approach (Phase 4) and the baseline classic FL method (Phase 2). The results demonstrate that FL games yield superior performance, achieving lower training loss and higher test accuracy, thereby highlighting their effectiveness in managing data heterogeneity.

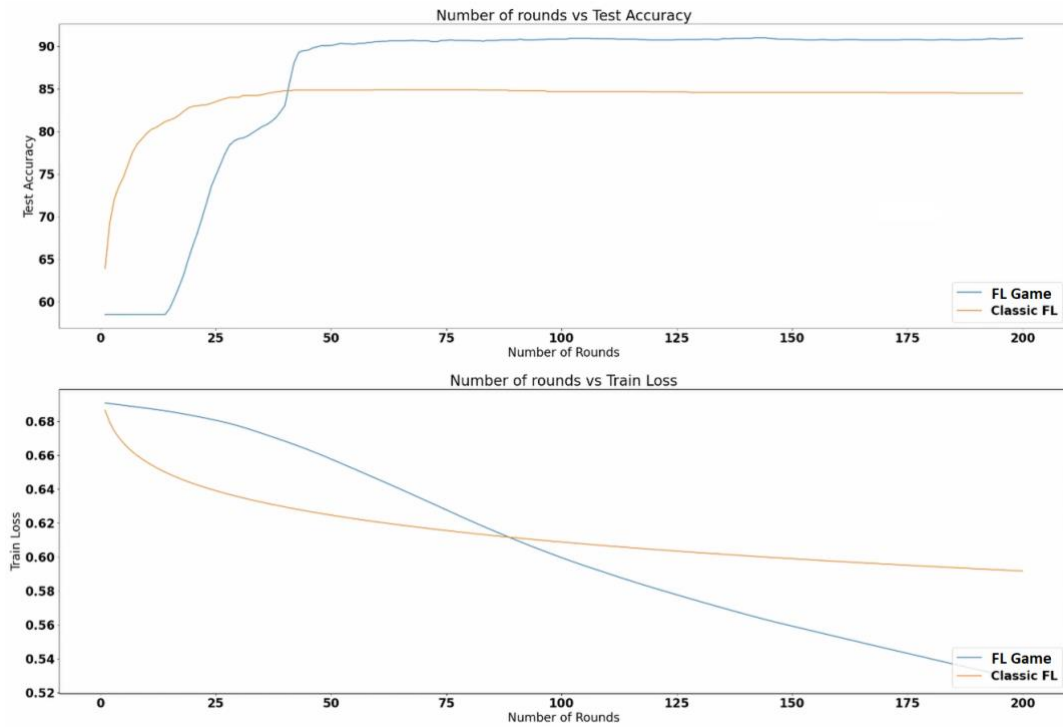


FIGURE 46 TRAINING ACCURACY AND TRAINING LOSS FOR CLASSIC FL AND FL GAMES WITH RIS DATA.

It is worth pointing out that report [74] provides additional details and complementary information of the FL implementation.

4.3.1.2 Implementation of algorithms within DESIRE6G architecture

For FL algorithms, the processes primarily involve

- Local training at edge nodes to local model update (FL clients)
- Local model aggregation to yield a global model (PS)

In this respect, the key components within architecture may be identified as follows:

- FL Clients: DUs, RUs, and RISs
- PS: Orchestrator (Near-RT RIC or Non-RT RIC) who acts as a parameter server for aggregating local models and managing FL training rounds.
- SMO: Supervises training policies and optimization related parameters, decisions.
- Data lake: Stores global models and possibly training parameters for AI/ML Models.

A step-by-step breakdown of message exchanges involved in the initialization, training, deployment, and inference phases of the FL algorithms is outlined in the following.

Initialization Phase

SMO / Non-RT RIC → FL Clients (mainly DUs):

- Distributes the initial global model W_0
- Sends FL parameters (e.g., training rounds, learning rate, local epoch size).

Local Training Phase (at DUs)

Each DU i :

- Uses its local dataset produced based on RUs-RISs-UEs channel estimation and rate data to train model W_t^i , where t denotes the epoch.
- Performs multiple (e.g., T) local epochs.

Model Upload Phase

DU i → SMO / Non-RT RIC:

- Sends locally trained model updates W_T^i .

Model Aggregation Phase (at PS)

SMO / Non-RT RIC:

- Aggregates received W_T^i s
- Stores the updated global model W_1

Global Model Broadcast Phase

SMO / Non-RT RIC → FL Clients (mainly DUs):

- Distributes the updated global model W_1
- Clients replace their local model with W_1 and prepare for the next round.

Convergence and Deployment

- Training continues for until the model converges (Checked by SMO / Non-RT RIC).
- Once completed, the final trained model is deployed into inference host (e.g., RIS).

We note that, even though the message flows have been identified within the DESIRE6G architecture, the FL algorithms discussed above have been demonstrated only as a PoC, see D5.2 [81].

4.3.2 Confidentiality Preserving Data Exchange with Third Party xApps

The analysis of data by a third party, with the adoption of rApps/xApps, enables ML as a Service (MLaaS) or cloud-based machine learning services. However, it also presents unique characteristics and challenges. The nature of the data whether it be images, text, numerical, real-time, streaming, or batch-processed- along with the criticality of response time require diverse solutions to ensure secure data transfer and analysis, preventing exposure during the process.

Data scrambling has been selected as a form of group-based anonymity for confidentiality preservation. This method strategically alters the arrangement of a dataset's rows and columns. In contexts where network data is involved, such as information from a telecommunications service provider, scrambling effectively conceals the specific details of signal degradation, including its location and timing. This ensures that the data remains confidential and incomprehensible to unauthorized third parties, while still preserving the ability to conduct statistical and analytical evaluations. Moreover, scrambling is a resource-efficient method that does not impose significant computational demands. Its lightweight nature allows for rapid data transfer and analysis, facilitating real-time processes. This is particularly advantageous when a third party must maintain online communication with the telecommunications provider to monitor the network continuously. This method has been preliminarily validated in an optical network scenario, where scrambled data are used as input features for soft failure detection based on unsupervised machine learning algorithm.

To detect soft failures in the data, we use unsupervised machine learning algorithms which do not need human guidance or beforehand training for data clustering [75] [76]. Data clustering can differentiate between the normal and different abnormal states of the network with grouping similar or proximate data points.

Figure 47 depicts the proposed approach. Following the chain from the left side, initially, the network state information dataset is collected by means of a Kafka-based monitoring framework [76]. The preprocessing phase involves reshaping the dataset into a format suitable for clustering algorithms, which requires time-series observations of all network elements. Each observation is documented as a single row in the dataset. Subsequently, scrambling is performed in the dataset to maintain confidentiality. All the steps are conducted at the network provider's premises.

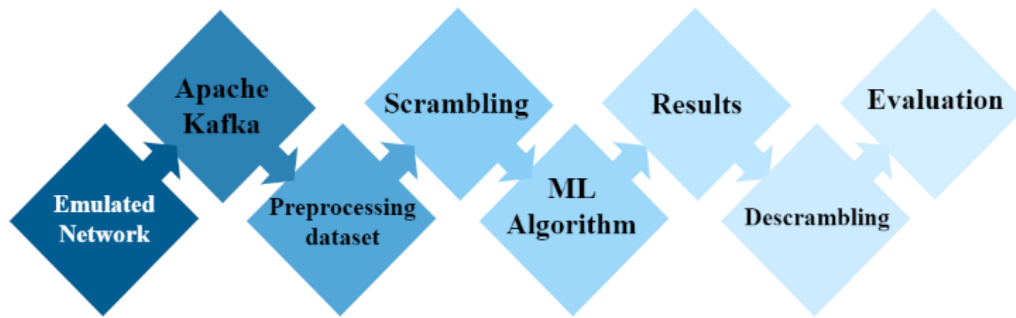


FIGURE 47 PROPOSED APPROACH FOR DATA EXCHANGE WITH A THIRD PARTY.

The scrambled dataset is then dispatched to a third party for failure detection analysis. The pivotal fifth block entails applying various clustering algorithms to the scrambled dataset. The third party executes this analysis and communicates the results back to the network provider.

The sixth block involves the transmission of the clusters of permuted results from the third party to the provider, who then retrieves the data in the seventh block to pinpoint failures. The provider, possessing knowledge of the network's states and failure specifics, decodes the results to identify the failures' locations.

Following Table 9 gives an example of data provided as input to our system.

TABLE 9 EXAMPLE: ADOPTED DATASET.

BER SPO1	BER SPO2	OSNR SPO1	OSNR SPO2	Amp1 Input Power	Amp1 Output Power	Amp2 Input Power	Amp2 Output Power	Amp3 Input Power	Amp3 Output Power	Amp4 Input Power	Amp4 Output Power
-16.8	-17.5	34.3	29.8	-11.3	3.9	-11.1	4	-12.7	2.6	-12.3	3
-16.8	-17.5	33.7	29.8	-11.2	3.9	-11.1	4	-12.7	2.6	-12.3	3
-16.8	-17.5	33.8	29.7	-11.3	3.9	-11.1	4	-12.7	2.6	-12.3	3

For clustering network states, we leverage the benefits of unsupervised machine learning algorithms, which do not require training with labeled data. This ensures that the third party analyzing the network never accesses the original, sensitive data, thereby enhancing confidentiality. We utilize Python scripts for the scrambling process and the Scikit Learn library for implementing clustering algorithms with initial setups being consistent for each algorithm on each dataset. The initial parameter for K -means is the number of clusters. DBSCAN needs two input parameters: eps which is the radius of the

neighborhood around each data point, and *min_sample*, the minimum number of points required to form a cluster. Similarly, HDBSCAN needs two input parameters; *min-cluster_size* which is the minimum number of points required to form a cluster, and *cluster-selection-epsilon* that is a distance threshold; clusters below this value will be merged. OPTICS needs one input parameter which is the minimum number of points to form a cluster as *min_samples*. Affinity propagation needs a pseudo-random number generator to control the starting state. Finally Spectral Clustering needs the dimension of the projection subspace as *n-clusters*.

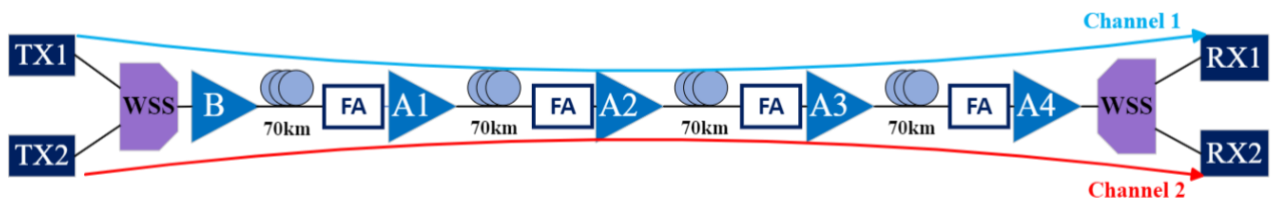


FIGURE 48 TESTBED SETUP.

The network scenario, illustrated in Figure 48, involves two transmit (TX) and two receive (RX) transponders, operating with different wavelengths (channel 1 at 193.1 THz and channel 2 at 193.2 THz, respectively) transmitting over the same four optical spans. To simulate various failure scenarios, Failure Actuators (FAs) are deployed across all spans. Each FA includes Wavelength Selective Switch (WSS) and can produce three distinct failure types on each span: a soft failure on channel 1, a soft failure on channel 2, and a soft failure on both channel 1 and channel 2. Each failure is programmed to manifest separately within its respective span, enabling the delineation of 12 unique failure states alongside a single state of normal operation. Failures are transient like the simple network. These conditions are expected to yield 13 discrete clusters, facilitating the subsequent application of clustering algorithms for analysis.

In our evaluation, the performance of the proposed workflow is compared with a benchmark workflow that does not implement scrambling. This comparison is crucial to determine how data scrambling impacts the clustering algorithms' effectiveness. Evaluating machine learning algorithms is crucial to identify the most effective one. However, due to the vast size of datasets and the multitude of clusters, manually verifying clustering results against true labels is impractical, often assessed by confusion matrix. To gauge the quality and validity of clustering outcomes, we employ both external and internal evaluation methods. External evaluations compare clusters to true labels, while internal evaluations examine the clusters' intrinsic structure. Given that different methods have varying assumptions and

limitations, and may not concur with the optimal clustering solution, it is vital to apply multiple evaluation metrics and interpret the results with discernment.

For external evaluations, we use Homogeneity, Completeness, V-measure, Adjusted Rand Index (ARI), and Adjusted Mutual Information (AMI). Homogeneity assesses if each cluster exclusively comprises data points from a single class, aiming for a perfectly homogeneous result where each cluster corresponds to one class label. Completeness evaluates whether all members of a class are grouped into a single cluster, with perfect completeness achieved when this criterion is met. V-measure is a harmonic mean of homogeneity and completeness, providing a singular metric. ARI measures the agreement between cluster assignments and class labels, based on the Rand Index, which considers the proportion of data point pairs correctly clustered. AMI quantifies the shared information between cluster assignments and class labels, reflecting the mutual dependence of the variables. The absolute value of these metrics ultimately ranges from 0 to 1, where higher values signify superior clustering quality. For internal evaluation, we utilize the Silhouette coefficient, which quantifies how well data points fit within their assigned clusters, see Table 10 and Table 11 for results.

TABLE 10 RESULTS FOR SCRAMBLED DATA.

Machine Learning Algorithm	DBSCAN	HDBSCAN	OPTICS	Kmeans	Affinity Propagation	Spectral Clustering
Homogeneity	0.926	0.901	0.873	0.821	0.992	0.739
Completeness	0.857	0.844	0.63	0.5	0.272	0.815
V-measure	0.89	0.872	0.732	0.622	0.427	0.775
Adjusted Rand Index	0.926	0.909	0.609	0.292	0.029	0.841
Adjusted Mutual Information	0.888	0.87	0.727	0.618	0.396	0.772
Silhouette Coefficient	0.705	0.715	-0.115	0.465	0.311	0.535
Average of Evaluations	0.865	0.852	0.614	0.553	0.405	0.746
Number of clusters	13	13	16	13	185	13
Number of noises	207	152	2999	0	0	0

TABLE 11 RESULTS FOR NON-SCRAMBLED DATA.

Machine Learning Algorithm	DBSCAN	HDBSCAN	OPTICS	Kmeans	Affinity Propagation	Spectral Clustering
Homogeneity	0.926	0.902	0.865	0.828	0.991	0.737
Completeness	0.857	0.845	0.616	0.505	0.269	0.811
V-measure	0.89	0.872	0.72	0.627	0.423	0.772
Adjusted Rand Index	0.926	0.91	0.595	0.295	0.033	0.841
Adjusted Mutual Information	0.888	0.87	0.715	0.623	0.388	0.769
Silhouette Coefficient	0.705	0.716	-0.105	0.471	0.273	0.588
Average of Evaluations	0.865	0.853	0.603	0.558	0.396	0.753
Number of clusters	13	13	16	13	218	13
Number of noises	207	149	2969	0	0	0

Overall, except for DBSCAN, all algorithm evaluations yield slightly lower scores in this more complex setting. The performance drop is particularly pronounced for *K*-means, suggesting it may not be suitable for complex datasets. OPTICS and Affinity Propagation continue to underperform, failing to accurately identify clusters. Among the remaining algorithms, Spectral Clustering shows improved results with scrambled data but is hindered by its time-intensive nature and inability to autonomously determine cluster numbers. Despite the close resemblance between DBSCAN and HDBSCAN, DBSCAN consistently outperforms, affirming its robustness in complex network scenarios.

From the standpoint of preserving confidentiality, the consistency of evaluation metrics between the original and scrambled datasets is a key indicator of effectiveness. As evidenced by Figure 49 and the accompanying tables, DBSCAN, HDBSCAN and Spectral clustering maintain high performance, suggesting they are better at finding clusters while even preserving data confidentiality during evaluation. This can be attributed to their density-based clustering approach for DBSCAN and HDBSCAN, which identifies clusters as regions of high density separated by low-density areas. Unlike *K*-means, which presumes convex-shaped clusters, DBSCAN's flexibility allows for clusters of any shape, making

it more suitable for our datasets. Importantly, data scrambling does not compromise the performance of these algorithms since it does not alter the underlying density and distribution of data points. Consequently, DBSCAN and HDBSCAN yield consistent results with both scrambled and unscrambled data, ensuring the confidentiality of the data and its owners.

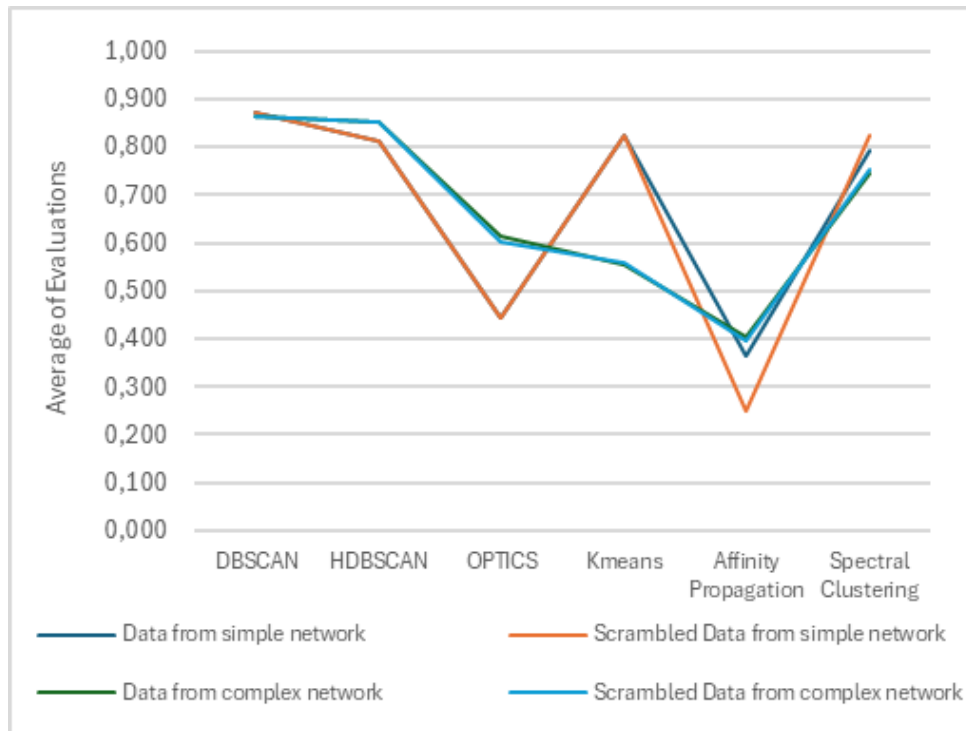


FIGURE 49 EVALUATIONS FOR DIFFERENT ALGORITHMS WITH DIFFERENT SENARIOS.

Identifying the correct number of clusters is crucial for detecting network failures, as it reflects the number of distinct failure regions. DBSCAN and HDBSCAN excel in this aspect, automatically determining the number of clusters with minimal input parameters, such as the neighborhood radius and the minimum number of points to form a cluster. Conversely, OPTICS and Affinity Propagation struggle to ascertain the correct cluster count, even with simpler datasets. *K*-means and Spectral Clustering, which require predefined cluster numbers, may not be viable when such information is unknown or variable.

4.3.3 Forecasting Element to Support 6G Operations

A significant aspect of edge computing is its role in minimizing latency through predictive resource allocation. As low-latency applications become increasingly prevalent, the ability to anticipate and

allocate resources before demand surges is paramount. Indeed, a proactive approach ensures optimal performance even during peak loads, catering to the needs of applications such as augmented reality, autonomous vehicles, and industrial IoT. By predicting potential bottlenecks or performance issues, the edge platform can proactively address these concerns, providing a reliable and consistent QoS for the defined use cases.

The Forecasting Element module aims at achieving this kind of behavior, while providing an extended flexibility in the type of resources that can be forecast. Such a module can be interfaced with the Management and Orchestration Layer and the monitoring system to provide forecast data useful for the service orchestration [77].

The module architecture, presented in Figure 50 offers a modular structure which can be used to address different forecasting jobs in parallel. It mainly interacts with the Kafka-based (or similar pub-sub frameworks) to retrieve metrics from one or more data streams (e.g., topics). The module then stores locally X measurements in the past to produce, through embedded AI/ML models, K measurements in the future ($X+K$).

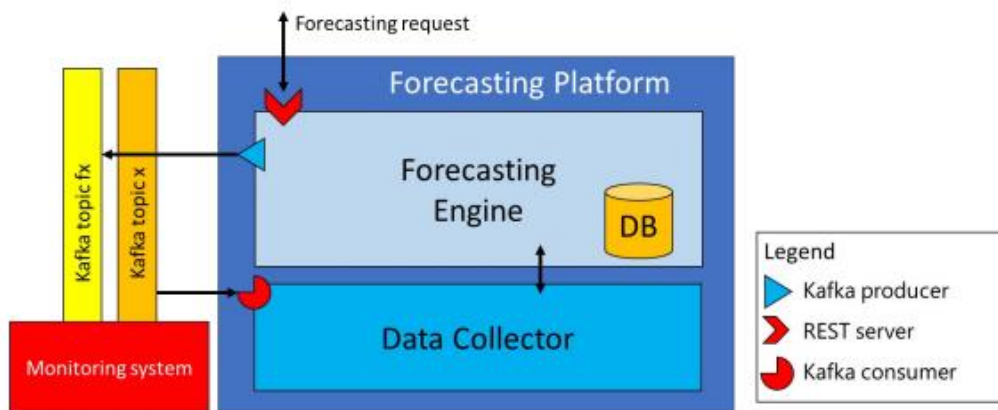


FIGURE 50 FORECASTING ELEMENT ARCHITECTURE.

More specifically, the main components are the following:

- **Data Collector:** Collecting metrics from the resources from the monitoring system.
- **Forecasting Engine:** Implementing four different sub functionalities listed below.
 - Handling the database where the data from the Data Collector is stored.
 - Running the different AI/ML models that can be used on the platform according to the input data streams configured for each job.
 - Exposing the forecast data to a dedicated topic/stream of each job.

- Gathering forecasting requests through an API server, which defines the jobs with input/output streams, input/output features and the model to be used (that can be trained or uploaded).

Considering the execution of a forecasting job at time t , Figure 51 shows the data structures in input and output manipulated by a forecasting job. Each forecasting job receives input from a matrix of data. Each row of the matrix represents one observation of the samples retrieved from the monitoring system, including one instance of all the values to be fed to the forecasting model. Each row has a size of N and includes the main feature (the one to be forecast), labelled with f in the figure, and the additional input features required by the model (i.e., f_1, f_2, f_{N-1}). The forecasting job temporarily stores last K observations (i.e., from t to $t - K$) and generates one forecasting value (f_t) referred to X steps ahead $t+X$.

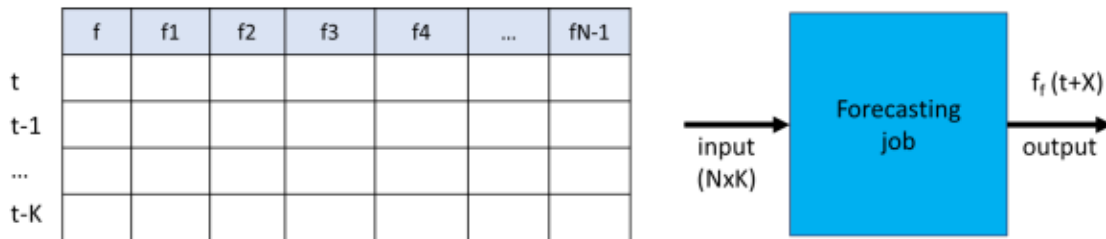


FIGURE 51 FORECASTING ELEMENT INPUT/OUTPUT LOGIC.

The module is completely written in Python, and allows different models to be used (e.g., LSTM-NNs, Gated-Recurrent Unit (GRU), and Convolutional Neural Networks (CNN)). These models can be then trained using different optimizers, such as Adam, and at a varying number of epochs.

The forecasting module has been adapted to work with K8s environment. In particular, it has been re-adapted to work with the metrics that are collected by the monitoring _manager component (Apache Kafka-based module for the collection of the relevant metrics from infrastructure and k8spods).

By considering a k8s cluster, a pod under test has been deployed. A user traffic emulator has been designed. The user emulator loads the pod with additional computation load, causing the 10% increase of the CPU utilization. The overall available features are the used cpu (millicore), the used Ram, the number of emulated users connected. The forecasting module has been tested with different input features (with and without the number of emulated users).

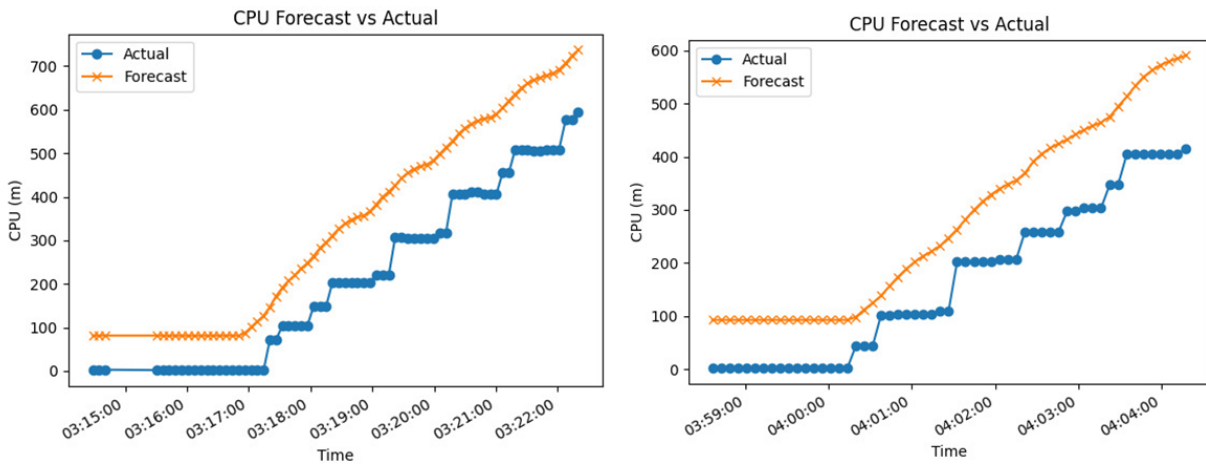


FIGURE 52. FORECASTING ELEMENT PERFORMANCE WITH K8S POD WITH (LEFT) AND WITHOUT (RIGHT) NUMBER OF EMULATED USERS.

Figure 52 shows the performance of the forecasting element in the two conditions (with and without the number of emulated users in the system). The forecasting module has been adopted to forecast the trend of the CPU load of the pod. The two plots show the trend of two curves, the actual CPU trend (represented in blue) and the forecasted one (represented in orange). In both cases is possible to highlight that the forecasting module is able to anticipate the behavior of the CPU load of the pod under test. This functionality is particularly relevant for one of the service assurance loops considered in Demo1, where the crucial pod related to the service can be deeply monitored, anticipating critical conditions, giving to the overall system the possibility, with some time guard, to activate pod replicas or to migrate the pod to a different server in the cluster.

4.3.4 Algorithms with Theoretical/Practical Insights

Some theoretical studies have also been conducted to characterize the performance of algorithms deployed in FL network settings from a signal processing perspective. Specifically, these studies focus on *i)* the impact of inexact communication [78], *ii)* the characterization of step size scheduling [79], and *iii)* the effects of graph optimization within a distributed FL framework [80]. A concise discussion is presented in section 5.3.4 of D3.2 [2]. In addition, a generic empirical study [82] is conducted to explore the potential of a class of machine learning algorithms based on multi-stage rollout methods [83], which offer a principled approach to make online sequential decisions in changing environments. The implications of these theoretical and empirical studies for the ongoing algorithmic developments within DESIRE6G remain to be further examined and are therefore left for future work.

5. Regulatory Aspects for Proposed AI/ML Approaches

The regulatory landscape for AI and ML is characterized by an emphasis on data privacy, ethical use, transparency, accountability, and the need for national and international cooperation in establishing guidelines. As these technologies continue to evolve, ongoing dialogue between stakeholders, including governments, industry leaders, and consumers, is essential to shape effective regulatory frameworks. Such frameworks could condition the assumptions and approaches currently taken in related research and developments, such as those in DESIRE6G. However, any relevant issue on this front in the project is not yet identified.

The regulatory aspects of AI and ML are evolving rapidly as governments and organizations look for addressing the challenges and implications associated with the deployment of these technologies.

The European Union (EU) has implemented the AI Act [84], which is considered the most advanced regulatory framework for AI so far. It includes several restrictions on large language models (LLMs) and mandates compliance and enforcement measures to ensure responsible AI use. The Act categorizes AI systems by risk levels, with stringent requirements for high-risk applications. These regulations focus on user consent and grant state agencies broad access to AI systems.

Key regulatory aspects considered so far are the following:

- **Regulation priorities:** Relevant regulatory priorities for AI include minimizing the impact on data privacy, ensuring that AI does not undermine democracy or human rights, and maintaining a “human in the loop” supervision for AI systems. These priorities indicate a growing demand for comprehensive regulatory frameworks that address ethical concerns and safeguard public interests.
- **Concerns Regarding Data Privacy:** Data privacy is a significant concern in the context of AI and ML. Regulations are expected to enforce strict guidelines on how personal data is collected, used, and shared, particularly in industries that rely heavily on consumer data. AI regulations often focus on ensuring that data used in training AI models complies with data privacy laws, including General Data Protection Regulation (GDPR) in Europe. This includes requirements for transparency in data usage and the implementation of adequate security measures to protect sensitive information. This involves ensuring that AI systems handle personal data appropriately, providing individuals with rights over their data, and minimizing any potential privacy breaches.

- **Impact of AI on Employment and Market Structure:** The introduction of AI and ML technologies raises concerns about their impact on employment within the telecom sector and other industries. However, in complex scenarios such as those proposed in DESIRE6G, AI and ML can be seen as an enabler rather than as a substitute technology, thus impacting positively in this regard.
- **Transparency and Accountability:** Regulations are likely to require organizations to be transparent about how AI systems operate and the data they use. This transparency is crucial for fostering trust among consumers and ensuring accountability in cases where AI systems cause harm or make biased decisions. Regulatory frameworks are increasingly addressing concerns about bias in AI models. Companies are required to conduct bias audits and ensure that their AI systems do not produce discriminatory outcomes. This includes developing frameworks that allow for the tracking and auditing of AI decisions and ensuring that AI systems can explain their actions and be held accountable for their outcomes
- **Ethics and Safety Considerations:** The ethical implications of AI, including bias in AI models and the potential for misuse, are critical regulatory concerns. There is a need for guidelines that ensure AI technologies are developed and implemented in ways that uphold ethical standards and prioritize safety. Within the context of DESIRE6G, this could be more related to the impact on specific use cases making use of DESIRE6G architecture and capabilities. This links with the notion of responsible AI. There is a strong emphasis on managing AI-related risks from ethical and legal perspectives. Organizations are expected to align with universal AI principles, comply with upcoming regulations, and demonstrate accountability for AI systems.
- **Risk-Based Approach:** The Act categorizes AI systems by risk levels, with stringent requirements for high-risk applications. It follows a comprehensive risk-based approach, classifying AI systems into four distinct categories [85]:
 - **Unacceptable Risk** (Prohibited Systems): Eight specific AI practices are banned, including harmful manipulation techniques, exploitation of vulnerabilities, social scoring systems, predictive policing based solely on profiling, untargeted facial recognition databases, emotion recognition in workplaces and educational institutions (with limited exceptions), biometric categorization for sensitive attributes, and real-time biometric identification in public spaces [86].

- **High-Risk Systems:** AI systems used in critical applications such as healthcare, education, employment, law enforcement, and border control face strict obligations including risk assessment and mitigation, high-quality datasets, logging capabilities, detailed documentation, human oversight measures, and accuracy requirements.
- **Limited Risk Systems:** These systems require specific transparency obligations, particularly when interacting with humans or generating content.
- **Minimal Risk Systems:** Most AI systems fall into this category with limited regulatory requirements, though they must still comply with general AI literacy requirements.

The regulatory landscape continues to evolve, and ongoing monitoring of regulatory developments will be essential to ensure DESIRE6G's compliance with emerging requirements and standards in the AI and telecommunications domains.

6. Conclusions

This document is the final version and an extension of D3.1 and D3.2 which tackles the implementation aspects of DESIRE6G intelligent and secure management, orchestration, and control. This document covers the project's progress from month 24 to month 32, building on Deliverable D3.2. It consolidates the architectural direction of the platform, provides detailed descriptions of its core functional components, and maps these components to the system-level requirements.

The specification of the End-to-End Service Management and Orchestration (SMO) framework was focused on the Service Orchestrator, the Optimization Engine, the Intent-Based Networking (IBN) manager, and the DLT-based federation enabler. Each component was presented with its internal architecture, interaction points, and implementation details, highlighting their interoperability and modularity within the overall platform.

In parallel, the deliverable covered the runtime mechanisms and operational aspects of service assurance and optimization, emphasizing the role of MAS-based deployment and secure agent-to-agent communication. The integration of privacy-preserving, and sustainable AI models highlights the platform's capacity to respond to evolving service demands and network dynamics, both in non-real-time and near-real-time domains.

Finally, the document provided a comprehensive alignment of the system components with the functional requirements defined in D2.1, ensuring full traceability and coverage of key design goals such as cloud-native deployment, cross-domain federation, AI-driven orchestration, and pervasive monitoring. This alignment validates the architectural coherence and implementation readiness of the platform.

7. References

- [1] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, "D3.1: Initial report on the DESIRE6G intelligent and secure management, orchestration, and control," 2023.
- [2] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, *D3.2: Interim report on the DESIRE6G intelligent and secure management, orchestration, and control*, 2024.
- [3] M. Ruiz, H. Shakespear-Miles, S. Barzegar, A. Sgambelluri and L. Velasco, "On the Benefits of Multi-Agent Systems for Operational Expenditure Savings," in *International Conference on Transparent Optical Networks (ICTON)*, 2025.
- [4] P. González, F. Alhamed, S. Barzegar, F. Paolucci, J. J. V. Olmos, M. Ruiz and L. Velasco, "Distributed Multi-Agent System fed with Telemetry Data for Near-Real-Time Service Operation," in *Optical Fiber Communications Conference and Exhibition (OFC)*, San Francisco, California, USA, 2024.
- [5] NUBIS, "Commit 6787c63," Opensource Community: kata-containers, 13 07 2023. [Online]. Available: <https://github.com/kata-containers/kata-containers/commit/6787c63900c6d4dc2de903a076f3230ecec4259b>. [Accessed 29 11 2024].
- [6] NUBIS, "containerd 1.7.0," Opensource Community: containerd, 10 03 2023. [Online]. Available: <https://github.com/containerd/containerd/releases/tag/v1.7.0>. [Accessed 29 11 2024].
- [7] NUBIS, "Commit 404c280," Opensource Community: Unikraft, 05 02 2024. [Online]. Available: <https://github.com/unikraft/unikraft/commit/404c280443c1caec7027924973c64a9aa3f78dd6>. [Accessed 29 11 2024].
- [8] NUBIS, "Commit e4eb664," Opensource Community: kata-containers, 02 06 2023. [Online]. Available: <https://github.com/kata-containers/kata-containers/commit/e4eb664d27f6cf7169c5d10f4383bdd8f3b892f6>. [Accessed 29 11 2024].
- [9] NUBIS, "runtime-rs: firecracker hypervisor backend #8070," Opensource Community: kata-containers, 26 09 2023. [Online]. Available: <https://github.com/kata-containers/kata-containers/pull/8070>. [Accessed 29 11 2024].

- [10] NUBIS, "runtime-rs: Fix QEMU backend for runtime-rs #10052," Opensource Community: kata-containers, 30 07 2024. [Online]. Available: <https://github.com/kata-containers/kata-containers/pull/10052>. [Accessed 29 11 2024].
- [11] NUBIS, "ci: Cleanup working dir when using self-hosted runners #9964," Opensource Community: kata-containers, 04 07 2024. [Online]. Available: <https://github.com/kata-containers/kata-containers/pull/9964>. [Accessed 29 11 2024].
- [12] NUBIS, "runtime-rs: Add pci_hotplug as a config option #9595," Opensource Community: kata-containers, 04 05 2024. [Online]. Available: <https://github.com/kata-containers/kata-containers/pull/9595>. [Accessed 29 11 2024].
- [13] NUBIS, "drivers/ukintctrl: Update GICv2 compatible list #1287," Opensource Community: kata-containers, 29 06 2024. [Online]. Available: <https://github.com/unikraft/unikraft/pull/1287>. [Accessed 29 11 2024].
- [14] NUBIS, "runtime-rs: Use vCPU and memory values from config #10435," Opensource Community: kata-containers, 18 10 2024. [Online]. Available: <https://github.com/kata-containers/kata-containers/pull/10435>. [Accessed 29 11 2024].
- [15] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, "D2.2: Functional Architecture Definition," 2024.
- [16] Fielding and R. Thomas, Architectural styles and the design of network-based software architectures, PhD Thesis, University of California, Irvine, 2000.
- [17] "fastapi," [Online]. Available: <https://github.com/fastapi/fastapi>. [Accessed 12 11 2024].
- [18] "RabbitMQ," [Online]. Available: <https://www.rabbitmq.com/>. [Accessed 12 11 2024].
- [19] N. Developers, "NetworkX v3.2.1 Documentation," [Online]. Available: <https://networkx.org/documentation/stable/>. [Accessed 28 08 2025].
- [20] S. H. J. L. S. K. L. Q.-L. H. a. Y. T. T. Wu, "A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, p. 11221136, 2023.

- [21] E. P. A. P. F. P. V. K. N. G. H. K. M. L. W.-t. Y. T. R. ". a. ". e. a. P. Lewis, "Retrievalaugmented Generation for Knowledge-intensive NLP Tasks," *Advances in Neural Information Processing Systems*, vol. 33, p. 9459–9474, 2020.
- [22] V. Lapascurt and I. Fiodorov, "Retrieval-augmented Generation Using Domain-specific Text: A Pilot Study," *Journal of Engineering Sciences*, vol. 2, pp. 48-59, 2024.
- [23] "User Group; Quality of ICT Services; Part 3: Template for Service Level Agreements (SLA)," ETSI, Tech. Rep. EG 202 009-3 V1.3.0, April 2015.
- [24] A. K. L. a. J. L. McClelland, "One-shot and Few-shot Learning of Word Embeddings," arXiv preprint arXiv:1710.10280, 2017.
- [25] ISO, "22400-2:2014 Automation systems and integration – Key performance indicators (KPIs) for manufacturing operations management".
- [26] 5G-ACIA, "Key 5G Use Cases and Requirements," <https://5gacia.org/whitepapers/key-5g-use-cases-and-requirements/>.
- [27] A. C. B. a. E. D. Biyar, "Intent-Based Management for Industrial Automation," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, Padova, Italy, 2024.
- [28] K. Antevski and C. J. Bernardos, "Federation in dynamic environments: Can blockchain be the solution?," *IEEE Communications Magazine*, vol. 60, no. 2, pp. 32-38, 2022.
- [29] A. Zahir, M. Groshev, K. Antevski, C. J. Bernardos, C. Ayimba and A. De La Oliva, "Performance evaluation of Private and Public Blockchains for multi-cloud service federation," *Proceedings of the 25th International Conference on Distributed Computing and Networking*, pp. 217–221, 2024.
- [30] S. Barzegar, M. Ruiz and L. Velasco, "Autonomous flow routing for near real-time quality of service assurance," *IEEE Transactions on Network and Service Management*, pp. 2504-2514, 2023.
- [31] P. González, A. Zahir, C. Grasselli, A. Muñoz, M. Groshev, S. Barzegar, F. Callegati, D. Careglio, M. Ruiz and L. Velasco, "Deployment of Secure Machine Learning Pipelines for Near-Real-Time Control of

- {6G} Network Services,” in *Optical Fiber Communications Conference and Exhibition (OFC)*, San Francisco, California, USA, 2024.
- [32] C. J. Bernardos and A. Mourad, ”RAW multidomain extensions,” SDO: Internet Engineering Task Force, [Online]. Available: <https://datatracker.ietf.org/doc/draft-bernardos-detnet-raw-multidomain/02/>. [Accessed 29 11 2024].
- [33] Nokia, Nokia Shanghai Bell, Ericsson, TELUS, KDDI, Verizon, Telefonica, Huawei, ”Rel18 CR TS28.541 Improve EP_Transport model to clarify connection point info,” SDO: 3GPP (SA5), 11 05 2023. [Online]. Available: https://www.3gpp.org/ftp/tsg_sa/WG5-TM/TSGS5_149/Docs/S5-234742.zip. [Accessed 29 11 2024].
- [34] L. M. Contreras, M. Boucadair, D. Lopez and C. J. Bernardos, ”An Evolution of Cooperating Layered Architecture for SDN (CLAS) for Compute and Data Awareness,” SDO: Internet Engineering Task Force, 23 05 2023. [Online]. Available: <https://datatracker.ietf.org/doc/draft-contreras-coinrg-clas-evolution/02/>. [Accessed 29 11 2024].
- [35] Ericsson, ”KI#2: New solutions: Providing measurement endpoint information to the 5GC/AF,” SDO: 3GPP (SA2), 15-19 04 2024. [Online]. Available: https://www.3gpp.org/ftp/tsg_sa/WG2_Arch/TSGS2_162_Changsha_2024-04/Docs/S2-2404482.zip. [Accessed 29 11 2024].
- [36] Telefonica, ”WG9-2024.001 Work Item Proposal – Inclusion of Transport in SMO (accepted and created),” SDO: O-RAN (WG1-WG9), [Online]. Available: <Not available publicly (file name: VZW.AO-2024.05.31-WG9-D-Transport-Inclusion-Data_Exposure)>.
- [37] V. M.Lacoste, ”Trusted Execution Environments for Telecom Industry, strengths, weaknesses, opportunities and threats,” *TEE Privacy and security Journal*, May 2023.
- [38] C. S.-H. Hsu, D. De Vleeschauwer and C. Papagianni, ”SLA Decomposition for Network Slicing: A Deep Neural Network Approach,” *IEEE Networking Letters*, 2023.
- [39] C. S.-H. Hsu, D. De Vleeschauwer and C. Papagianni, ”Online SLA Decomposition: Enabling Real-Time Adaptation to Evolving Systems,” arXiv preprint arXiv:2408.08968, 2024.

- [40] A. D. P. G. C. P. Cyril Shih-Huan Hsu, "Transformer-Empowered Actor-Critic Reinforcement Learning for Sequence-Aware Service Function Chain Partitioning," *Transactions on Network Science and Engineering (TNSE)*, 2025.
- [41] J. Wang, J. Liu, J. Li and N. Kato, "Artificial intelligence-assisted network slicing: Network assurance and service provisioning in 6G," *IEEE Vehicular Technology Magazine*, vol. 18, no. 1, pp. 49-58, 2023.
- [42] D. De Vleeschauwer, C. Papagianni and A. Walid, "Decomposing SLAs for network slicing," *IEEE Communications Letters*, vol. 25, no. 3, pp. 950-954, 2020.
- [43] R. Su, D. Zhang, R. Venkatesan, Z. Gong, C. Li, F. Ding, F. Jiang and Z. Zhu, "Resource allocation for network slicing in 5G telecommunication networks: A survey of principles and models," *IEEE Network*, vol. 33, no. 6, pp. 172-179, 2019.
- [44] M. Iannelli, M. R. Rahman, N. Choi and L. Wang, "Applying machine learning to end-to-end slice SLA decomposition," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, Virtual Conference, 2020.
- [45] Y. L. D. L. a. V. C. G. Sun, "Service function chain orchestration across multiple domains: A full mesh aggregation approach," 2018.
- [46] M. R. a. S. Schmid, "On the hardness and inapproximability of virtual network embeddings," *IEEE/ACM Transactions on Networking*, 2020.
- [47] C. S.-H. Hsu, A. Dalgikitsis, C. Papagianni and P. Grosso, "Transformer-Empowered Actor-Critic Reinforcement Learning for Sequence-Aware Service Function Chain Partitioning," arxiv, 2025.
- [48] P. T. A. Quang, A. Bradai, K. D. Singh, G. Picard and R. Riggio, "Single and multi-domain adaptive allocation algorithms for VNF forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 98-112, 2018.
- [49] D. Dietrich, A. Abujoda, A. Rizk and P. Papadimitriou, "Multi-Provider Service Chain Embedding With Nestor," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 91-105, 2017.

- [50] J. Li, W. Shi, H. Wu, S. Zhang and X. Shen, "Cost-Aware Dynamic SFC Mapping and Scheduling in SDN/NFV-Enabled Space-Air-Ground-Integrated Networks for Internet of Vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 8, pp. 5824-5838, 2022.
- [51] H. Moens and F. D. Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *10th International Conference on Network and Service Management*, Rio de Janeiro, Brazil, 2014.
- [52] T. Nguyen, C. Pham and A. Nguyen, "Deep reinforcement learning for availability-aware service function chain placement," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2543-2556, 2021.
- [53] D. D. V. C.-Y. C. K. D. S. P. P. Angelos Pentelas, "Deep Multi-Agent Reinforcement Learning With Minimal Cross-Agent Communication for SFC Partitioning".*IEEE Access*.
- [54] X. Yang, S. Gao and H. Wang, "Graph neural networks for service function chain partitioning," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1567-1579, 2022.
- [55] L. G. P.-V. M. K. R. L. A. C. V. Anestis Dalgkitis, "SCHEMA: Service Chain Elastic Management with Distributed Reinforcement Learning," in *GlobeCom*, Madrid, Spain, 2021.
- [56] P.-V. M. A. A. G. K. C. V. Anestis Dalgkitis, "Dynamic Resource Aware VNF Placement with Deep Reinforcement Learning for 5G Networks," in *GlobeCom*, Taipei, Taiwan, 2020.
- [57] J. R. Rice, "The algorithm selection problem," *Advances in Computers*, vol. 15, pp. 65-188, 19 February 1975.
- [58] Y. M. M. S. a. K. T. S. Kadioglu, "ISAC – Instance-Specific Algorithm Configuration," in *19th European Conference on Artificial Intelligence (ECAI 2010)*, 2010.
- [59] C. S. a. A. C. d. C. T. Cunha, "A label ranking approach for selecting rankings of collaborative filtering algorithms," in *33rd Annual ACM Symposium on Applied Computing*, 2018.
- [60] Y. Z. J. W. a. K. T. X. Wu, "AS-LLM: When algorithm selection meets Large Language Model," 2024.
- [61] M. A. L. K. a. H. H. Damir Pulatov, "Opening the black box: Automated software analysis for algorithm selection," in *International Conference on Automated Machine Learning*, 2022.

- [62] C. S.-H. Hsu, J. Martín-Pérez, D. De Vleeschauwer, K. Kondepu, L. Valcarengi, X. Li and C. Papagianni, "A Deep RL Approach on Task Placement and Scaling of Edge Resources for Cellular Vehicle-to-Network Service Provisioning," arXiv preprint arXiv:2305.09832, 2023.
- [63] C. S.-H. Hsu, J. Martín-Pérez, C. Papagianni and P. Grosso, "V2N service scaling with deep reinforcement learning," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Miami, FL, USA, 2023.
- [64] A. Medeiros, A. Neto, S. Sampaio, R. Pasquini and J. Baliosian, "Enabling elasticity control functions for cloud-network slice-defined domains," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Budapest, Hungary, 2020.
- [65] H. Jiang, X. Dai, Z. Xiao and A. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 7, pp. 4000-4015.
- [66] S. Liu, J. a. Z. C. Tian and T. Li, "Joint computation offloading and resource allocation in vehicular edge computing networks," *Digital Communications and Networks*, 2022.
- [67] S. Li, N. Zhang, R. Jiang, Z. Zhou, F. Zheng and G. Yang, "Joint task offloading and resource allocation in mobile edge computing with energy harvesting," *Journal of Cloud Computing*, vol. 11, no. 1, p. 17, 2022.
- [68] P. Lai, Y. Tao, J. Qin, Y. Xie, S. Zhang, S. Tang, Q. Huang and S. Liao, "Joint optimization of application placement and resource allocation for enhanced performance in heterogeneous multi-server systems," *Computer Networks*, vol. 253, p. 110692, 2024.
- [69] J. Chen, H. Xing, Z. Xiao, L. Xu and T. Tao, "A DRL agent for jointly optimizing computation offloading and resource allocation in MEC," *IEEE Internet of Things Journal*, vol. 8, no. 24, pp. 17508-17524, 2021.
- [70] D. De Vleeschauwer, J. Baranda, J. Mangues-Bafalluy, C. F. Chiasserini, M. Malinverno, C. Puligheddu, L. Magoula, J. Martín-Pérez, S. Bampounakis, K. Kondepu and Others, "5Growth data-driven AI-based scaling," in *Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 2021.

- [71] J. Martín-Pérez, K. Kondepudi, D. De Vleeschauwer, V. Reddy, C. Guimaraes, A. Sgambelluri, L. Valcarengi, C. Papagianni and C. J. Bernardos, "Dimensioning V2N services in 5G networks through forecast-based scaling," *IEEE Access*, vol. 10, pp. 9587-9602, 2022.
- [72] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, "D5.1: Preliminary experimental setup and data set collection," 2024.
- [73] C. B. Chaaya, S. S and B. M, "Federated Learning Games for Reconfigurable Intelligent Surfaces via Causal Representations," in *IEEE Global Communications Conference*, Kuala Lumpur, Malaysia, 2023.
- [74] C. Mawela, A web-based solution for Federated learning with LLM based Automation, University of Oulu Repository, MSc Thesis, The University of Oulu, 2024.
- [75] A. Yeganehfallah, A. Sgambelluri, A. Pacini, L. Valcarengi and M. F. Silva, "Effectiveness of Confidentiality-Preserving Clustering Algorithms for Soft Failure Detection in Optical Networks," in *International Conference on High Performance Switching and Routing (HPSR)*, Pisa, Italy, 2024.
- [76] M. F. Silva, A. Sgambelluri, A. Pacini, F. Paolucci, A. Green, D. Mascarenas and L. Valcarengi, "Confidentiality-preserving machine learning algorithms for soft-failure detection in optical communication networks," *Journal of Optical Communications and Networking*, vol. 15, no. 8, pp. C212-C222, 2023.
- [77] L. C. P. Valcarengi, A. Sgambelluri, E. Paolini and A. Pacini, "A Flexible Forecasting Platform Enabling Zero Touch Networking and Digital Twinning," in *International Conference on Transparent Optical Networks (ICTON)*, Bucharest, Romania, 2023.
- [78] S. Ranaweera, C. Weeraddana, P. Dharmawansa and C. Fischione, "On the Convergence of Inexact Gradient Descent With Controlled Synchronization Steps," *IEEE Signal Processing Letters*, vol. 30, pp. 703-707, 2023.
- [79] H. Abeynanda, C. Weeraddana and C. Fischione, "On the Characteristics of the Conjugate Function Enabling Effective Dual Decomposition Methods," arXiv preprint arXiv:2405.08933, 2024.

- [80] T. AlShammari, C. Weeraddana and M. Bennis, "BayGO: Decentralized Bayesian Learning and Information-Aware Graph Optimization Framework," *IEEE Transactions on Signal Processing*, 2024.
- [81] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, "D5.2: Report on Evaluation Results and Initial Proof-of-Concept Demonstrations," 2025.
- [82] A. Vivekananthan, Adaptive autonomy in multi-agent systems with modeled interactions : an empirical analysis, University of Oulu Repository, MSc Thesis, The University of Oulu, 2025.
- [83] D. Bertsekas, Rollout, policy iteration, and distributed reinforcement learning, Athena Scientific, 2021.
- [84] European Union, "EU Artificial Intelligence Act," 2024. [Online]. Available: <https://artificialintelligenceact.eu/ai-act-explorer/>. [Accessed 13 11 2024].
- [85] European Union, "AI Act," 01 08 2025. [Online]. Available: <https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai>.
- [86] European Union, "Commission publishes the Guidelines on prohibited artificial intelligence (AI) practices, as defined by the AI Act," 31 07 2025. [Online]. Available: <https://digital-strategy.ec.europa.eu/en/library/commission-publishes-guidelines-prohibited-artificial-intelligence-ai-practices-defined-ai-act>.
- [87] K. Antevski and C. Bernardos, "Federation of 5G services using distributed ledger," *Wiley Special Issue*.
- [88] L. Velasco and e. al, "Securing Multi-Agent Systems for the Near-Time Control of 6G Services," in *EUCNC 2023*.
- [89] O. Aluko and A. Kolonin, "Proof of reputation: An alternative consensus mechanism for blockchain systems," in *INS4*, 2021.
- [90] N. Jiang and e. al, "Reputation-driven Dynamic Node Consensus and Reliability Sharding Model in IoT Blockchain," in *Algorithms*, 2022.

- [91] C. Shih-Huan Hsu, D. De Vleeschauer and C. Papagianni, "SLA Decomposition for Network Slicing: A Deep Neural Network Approach," *IEEE Networking Letters* [under submission].
- [92] J. Wang, J. Liu, J. Li and N. Kato, "Artificial intelligence-assisted network slicing: Network assurance and service provisioning in 6g," *IEEE Vehicular Technology Magazine*, vol. 18, no. 1, pp. 49-58, 2023.
- [93] R. B. Uriarte and e. al., "Blockchain-Based Decentralized Cloud/ Fog Solutions: Challenges, Opportunities, and Standards," *IEEE Communications Standards Magazine*, vol. 2, no. 3, pp. 22-28, 2018.
- [94] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, "D3.1: Initial report on the intelligent and secure management, orchestration, and control platform," 2023.
- [95] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, "D2.1: Definition of Use Cases, Service Requirements and KPIs/KVIs," 2023.
- [96] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, "D4.2: Interim report on the unified programmable data plane layer of DESIRE6G," 2024.
- [97] DESIRE6G, HEU 6G SNS JU Project Grant No. 101096466, "D3.3: Final report on the DESIRE6G intelligent and secure management, orchestration, and control," 2025.
- [98] User Group, Quality of telecom services (Part 1), "Methodology for identification of indicators relevant to the Users," ETSI EG 202 009-1 V1.3.1 , 2014-12 .
- [99] B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *Artificial intelligence and statistics (PMLR)*, pp. 1273-1282, 2017.
- [100] A. Martin, L. Bottou, I. Gulrajani and D. Lopez-Paz, "Invariant risk minimization," arXiv preprint arXiv:1907.02893, 2019.
- [101] C. B. Issaid, S. Samarakoon, M. Bennis and H. V. Poor, "Federated distributionally robust optimization for phase configuration of RISs," in *IEEE Global Communications Conference (GLOBECOM)*, Madrid, Spain, 2021.

- [102] NEC Laboratories, "SOL v0.6.1 Documentation," [Online]. Available: <https://sol.neclab.eu/docs/v0.6/releases/v0.6.html>. [Accessed 12 11 2024].
- [103] M. Rost and S. Schmid, "On the hardness and inapproximability of virtual network embeddings," *IEEE/ACM transactions on networking*, vol. 28, no. 2, pp. 791-803, 2020.
- [104] J. R. Rice, The Algorithm Selection Problem, Academic Press, 1976.
- [105] R. J. M. S. a. A. E. D. Bhamare, "A survey on service function chaining," *Journal of Network and Computer Applications*, 2016.
- [106] C. P. a. P. G. C. S.-H. Hsu, "RAILS: Risk-aware iterated local search for joint SLA decomposition and service provider management in multi-domain networks," *IEEE 26th International Conference on High Performance Switching and Routing (HPSR)*, 2025.
- [107] 5G-SMART, "Analysis of Business Value Creation Enabled by 5G for Manufacturing Industries," <https://5gsmart.eu>.
- [108] NetworkX, "NetworkX v3.2.1 Documentation," [Online]. Available: <https://networkx.org/documentation/stable/>. [Accessed 28 08 2025].
- [109] A. Pentelas, D. De Vleeschauwer, C.-Y. Chang, K. De Schepper and P. Papadimitriou, "Deep multi-agent reinforcement learning with minimal cross-agent communication for sfc partitioning," *IEEE Access*, vol. 11, pp. 40384-40398, 2023.
- [110] X. Wu, Y. Zhong, J. Wu, B. Jiang and C. Tan, "Large Language Model-Enhanced Algorithm Selection: Towards Comprehensive Algorithm Representation".
- [111] H. R. Lourenco, O. C. Martin and T. Stutzle, "Iterated Local Search. Bosto," 2003.