

SAFE-6G

A Smart and Adaptive Framework for Enhancing Trust in 6G Networks

Deliverable D4.6: User-centric Resilience Trust Function

Date: 30/09/2026

Version: V1.0

DISCLAIMER

This document contains confidential information reserved for the partners of the SAFE-6G (“A Smart and Adaptive Framework for Enhancing Trust in 6G Networks”) Consortium and is subject to the confidentiality obligations set out in the Grant Agreement 101139031 and to the SAFE-6G Consortium Agreement.

This document contains information, which is proprietary to the SAFE-6G Consortium that is subject to the rights and obligations and the terms and conditions applicable to the Grant Agreement. The action of the SAFE-6G Consortium is funded by the European Commission.

Neither this document nor the information contained herein shall be used, copied, duplicated, reproduced, modified, or communicated by any means to any third party, in whole or in parts, except with prior written consent of the SAFE-6G Consortium. In such case, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced. In the event of infringement, the consortium reserves the right to take any legal action it deems appropriate.

This document reflects only the authors’ view and does not necessarily reflect the view of the European Commission. Neither the SAFE-6G Consortium as a whole, nor a certain party of the SAFE-6G Consortium warrant that the information contained in this document is suitable for use, nor that the use of the information is accurate or free from risk, and accepts no liability for loss or damage suffered by any person using this information.

The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Grant Agreement	101139031
Document number	D4.6
Document title	User-centric Resilience Trust Function
Lead Beneficiary	TID
Editor(s)	Miguel Posada (TID) Pelayo Torres (TID)
Author(s)	Miguel Posada (TID) Pelayo Torres (TID)
Dissemination level	Public
Contractual date of delivery	30/09/2026
Status	Final
File name	SAFE-6G_D4.6_V1.0.pdf

Revision History

Version	
V0.1	Initial ToC
V0.2	First full draft of Sections 1 through 9 and the Annex, based on the WP4 reference architecture and the project's implementation.
V0.3	Editorial corrections against the current implementation; consolidated Section 6; added the message-contract table (Section 4) and the nApp rule-catalogue table (Annex 1); added the Glossary, Executive Summary, and Keywords.
V0.8	Version ready for internal review, made by BULL and IQBT
V1.0	Final version ready for submission

GLOSSARY

Abbreviations/Acronym	Description
5QI	5G QoS Identifier
A2A	Agent-to-Agent (protocol)
aLoTW	Achievable Level of Trustworthiness
API	Application Programming Interface
ARP	Allocation and Retention Priority
CoCo	Cognitive Coordinator
cLoTW	Calibrated Level of Trustworthiness
DoA	Description of Action
gNB	next generation Node B
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
KPI	Key Performance Indicator
KVI	Key Value Indicator
LoTW	Level of Trustworthiness
MLOps	Machine Learning Operations
MSISDN	Mobile Station International Subscriber Directory Number
nApp	Network Application
NEF	Network Exposure Function
NFV	Network Function Virtualisation
nLoTW	Non-calibrated Level of Trustworthiness
OpenCAPIF	Open-source implementation of the Common API Framework
PRB	Physical Resource Block
QoS	Quality of Service
REST	Representational State Transfer
RRC	Radio Resource Control
SDK	Software Development Kit
SSC	Session and Service Continuity
SUPI	Subscription Permanent Identifier
TF	Trust Function
UE	User Equipment
UPF	User Plane Function
vApp	Vertical Application
XR	Extended Reality

EXECUTIVE SUMMARY

This deliverable defines the User-centric Resilience Trust Function, one of the five Trust Functions of the SAFE-6G trustworthiness framework, at the conceptual and architectural level. It describes the function's role within SAFE-6G, the trust signals it exchanges with the Cognitive Coordinator (CoCo), and its distribution across three cooperating layers, the local-ai-agent, the vApp, and the nApp, which respectively assess the achievable resilience trust level for a user equipment (UE) session, orchestrate session-level monitoring and decisions, and enforce the resulting configuration on the core network. The deliverable explains how resilience is evaluated and enforced and identifies future research pathways for extending the function beyond the scope of the current implementation. Deployment of measurements, performance results, and the design or training of the underlying artificial intelligence and machine learning models are outside the scope of this document.

KEYWORDS

Resilience, Trust Function, Trustworthiness, Cognitive Coordinator, User-centric, 6G

TABLE OF CONTENTS

1	<i>Introduction</i>	1
1.1	Purpose and scope of the deliverable	1
1.2	Relation to other deliverables and work packages	2
2	<i>Related Work</i>	4
2.1	Resilience in mobile networks: from network-wide redundancy to per-session recovery	4
2.2	Resilience mechanisms in the 3GPP 5G system	4
2.3	Self-organizing and self-healing network management	5
2.4	Resilience in virtualised, cloud-native infrastructure	5
2.5	User-centric and AI-driven resilience control	5
2.6	Gap addressed by the SAFE-6G Resilience Trust Function	5
3	<i>Trust Function Definition & Role in SAFE-6G</i>	6
3.1	Definition of the Resilience Trust Function	6
3.2	Trustworthiness signals: inputs and outputs	6
3.3	Interaction with the Cognitive Coordinator	7
3.4	Interaction boundaries with the other Trust Functions	8
4	<i>Reference Architecture</i>	9
4.1	Architectural overview and design principles	9
4.2	Component decomposition	10
4.3	Message-driven interaction model	11
4.4	Topic catalogue and message schemas	12
4.5	Deployment view	12
4.6	End-to-end interaction	13
4.7	Architectural assumptions and constraints	14
4.8	Session state and message correlation	15
5	<i>Trust Function Layers</i>	17
5.1	Local AI Agent	17
5.1.1	Role and position in the resilience workflow	17
5.1.2	Implementation stack	17
5.1.3	Inputs and message contract	17
5.1.4	Preprocessing and telemetry	18
5.1.5	Decision-making approach	18

5.1.6	Outputs and message contract	23
5.1.7	Limitations.....	24
5.1.8	Failure handling.....	24
5.2	Vertical Application (vApp)	24
5.2.1	Role and position in the resilience workflow	24
5.2.2	Implementation stack	24
5.2.3	Inputs and message contract	25
5.2.4	Preprocessing and telemetry	25
5.2.5	Decision-making approach.....	25
5.2.6	Outputs and message contract	28
5.2.7	Limitations.....	29
5.2.8	Failure handling.....	29
5.3	Network Applications (nApp)	29
5.3.1	Role and position in the resilience workflow	29
5.3.2	Implementation stack	29
5.3.3	Inputs and message contract	29
5.3.4	Decision and enforcement Logic.....	30
5.3.5	Outputs and message contract	31
5.3.6	Limitations.....	31
5.3.7	Failure handling.....	31
6	Trust Evaluation & Enforcement Logic	32
6.1	Data acquisition and indicator calculation	32
6.2	Trust evaluation	32
6.3	Enforcement logic.....	32
6.4	Feedback, reassessment, and fallback handling	33
6.5	Telemetry quality and timing	33
6.6	Verification of execution and service effect	34
6.7	Illustrative session workflow.....	35
7	Future Research Pathways	36
7.1	Evolution of the closed-loop resilience workflow.....	36
7.2	Advanced intelligence and user-aware resilience.....	36
7.3	Architectural evolution and interoperability.....	37
7.4	Validation, scalability, and operationalisation	37
7.5	Application-aware resilience and interaction with the Metaverse environment	37
8	Conclusions	39
9	REFERENCES.....	40

List of FIGURES

Figure 1: High-level architecture of the User-centric Resilience Trust Function	10
Figure 2: Resilience Trust Function session lifecycle	14
Figure 3: MLOps workflow for synthetic data generation, model training, and deployment.....	19

List of TABLES

Table 1: Level of Resilience levels, flavours and their enforcement intent.....	7
Table 2: Message Contracts	12
Table 3: Logical states for resilience session validation.....	16
Table 4: Variables of the Knowledge Seed.....	20
Table 5: Frequency Band Ranges	20
Table 6: Occupancy Ranges.....	20
Table 7: UPF Load Ranges	21
Table 8: ALoTW Endpoints	21
Table 9: Results of ALoTW Models Testing.....	22
Table 10: ALoTW Models Evaluation	22
Table 11: Reinforcement Learning Models Endpoints.....	27
Table 12: Results of Reinforcement Learning Models Testing	27
Table 13: Action model endpoints.....	30

1 INTRODUCTION

1.1 PURPOSE AND SCOPE OF THE DELIVERABLE

The User-centric Resilience Trust Function (hereafter, the Resilience Trust Function) is one of five Trust Functions (TFs) that constitute the trustworthiness framework of SAFE-6G, a Smart and Adaptive Framework for Enhancing Trust in 6G Networks, alongside the Safety, Security, Privacy, and Reliability Trust Functions. The Resilience Trust Function addresses the resilience dimension of trustworthiness: the capacity of a service, session, or network configuration to withstand, adapt to, and recover from degradations, faults, or unexpected operating conditions while remaining aligned with user intent. [1]

Within Work Package 4 (WP4), "Trustworthiness framework," the Resilience Trust Function operates under the coordination of the Cognitive Coordinator (CoCo), which converts high-level user intents into a Level of Trustworthiness (LoTW) and its calibrated form, the Calibrated Level of Trustworthiness (cLoTW), across the five trust dimensions. For each User Equipment (UE) session, the Resilience Trust Function evaluates the Achievable Level of Trustworthiness (aLoTW): the resilience trust level that can practically be delivered given current network conditions. It reports this value to CoCo and, in turn, receives a resolved resilience trust target that it uses to select and enforce appropriate network configurations. [1]

The Resilience Trust Function is architecturally distributed across three cooperating layers, consistent with the common WP4 reference design: the local-ai-agent, responsible for assessing achievable resilience for a UE session; the vertical application (vApp), responsible for orchestrating resilience-related decisions at the session level in a user-centric manner; and the network application (nApp), responsible for translating those decisions into core network configuration actions. [1]

The function makes resilience assessments and configuration decisions for a specific user session, taking account of the requested trust target, device type and current network conditions. This adds a SAFE-6G coordination layer to existing continuity, QoS and infrastructure-recovery mechanisms; it does not replace those mechanisms or imply that established mobile-network standards lack per-UE capabilities. The intended operational benefit is to reduce the effect of degradation on the selected session through coordinated, context-dependent reconfiguration [1], [3].

The objective of Deliverable D4.6 is to specify the Resilience Trust Function and its implementation approach: its role within SAFE-6G, the trust signals exchanged with CoCo, the responsibilities of its three layers, the models supporting its decisions, and the translation of approved actions into core-network updates. It also relates this design to the requirements and validation framework established in D2.1 [2] and D2.4 [2], [6].

The scope includes conceptual and architectural design, message and API interactions, model configuration and synthetic evaluation, enforcement behaviour, and the conditions needed for integration validation. The implementation descriptions refer to the three component repositories listed in [7]–[9].

System-level deployment measurements, field performance and demonstrated KPI achievement are outside the scope of this deliverable and addressed in WP5. The internal models and reconciliation logic of CoCo belong to the associated WP4 deliverables.

The main contributions of this deliverable are fourfold. First, it defines the responsibility boundary of the User-centric Resilience Trust Function within the SAFE-6G trustworthiness framework. Second, it specifies the message contracts and operational sequence connecting CoCo, the local-ai-agent, the vApp and the nApp. Third, it documents the model-serving and deterministic enforcement mechanisms implemented for session-level resilience control. Fourth, it identifies the evidence still required to validate that an approved configuration change produces a measurable resilience improvement for the affected UE session.

The remainder of this deliverable is structured as follows. Section 2 situates the Resilience Trust Function with respect to relevant background on resilience and trust in 6G networks. Section 3 formally defines the Resilience Trust Function and its role within SAFE-6G. Section 4 presents the reference architecture. Section 5 describes, in turn, the local-ai-agent, vApp, and nApp layers, also describes the decision models used by the Resilience Trust Function and reports their component-level tests; those results are not evidence of end-to-end service recovery or of generalisation to operational network data.

Section 6 explains the trust evaluation and enforcement logic. Section 7 outlines future research pathways building on the architecture and implementation presented in this deliverable. Section 8 concludes the deliverable.

1.2 RELATION TO OTHER DELIVERABLES AND WORK PACKAGES

D4.6 connects the WP4 architecture in D4.1 [1] to the requirements in D2.1 [2], Section 5.7.2 [2], and the evaluation framework in D2.4 v1.1 [6]. The following traceability assessment uses the thirteen resilience requirement identifiers in D2.1 [2]. Coverage describes the design presented here; it is not a declaration that integration acceptance tests have passed.

- REQ-FRES-FRA-F-M-01 (framework integration): health-check endpoints and trust-message handling provide part of the required integration surface. Exposure of deployment location, resource availability and resilience-data storage needs separate verification (Sections 4.5 and 5).
- REQ-FRES-FRA-F-M-02 (trust-level consumption): Kafka messages carry nLoTW and cLoTW. The HTTP interface wording in D2.1 [2] requires an agreed interface mapping or an explicit acceptance of this architectural deviation (Section 4.4).
- REQ-FRES-DATA-F-M-03 (user intent): CoCo interprets user intent and passes the resilience target to the function. Validation should trace an intent through to the corresponding session decision (Section 3.4).
- REQ-FRES-DATA-F-M-04 (MLOps-driven action): the score and action predictors inform decisions, and the nApp applies the resulting configuration. Effectiveness remains subject to integration validation (Sections 5 and 6).

- REQ-FRES-NET-F-M-05 (core redundancy control): slice and UPF selection are described, but redundancy management across AMF, UPF and SMF is not established by these operations alone (Section 5.3).
- REQ-FRES-FRA-F-M-06 (function information): a dedicated interface for deployment location and resource availability is not specified in the current design. A health endpoint alone does not provide this information.
- REQ-FRES-INFRA-F-M-07 (infrastructure redundancy): this depends on the continuum deployment. Single replicas and disabled autoscaling do not establish redundant execution of the three services (Section 4.5).
- REQ-FRES-NET-NF-M-08 (secure CAPIF interaction): OpenCAPIF supports discovery and authorisation. Verification must also cover the configured authentication and transport protections (Section 4.9).
- REQ-FRES-NET-NF-M-09 (OpenCAPIF integration): the three components use OpenCAPIF for CumuCore access. Provider publication, invoker onboarding and the selected API versions should be included in the integration evidence (Sections 4 and 5).
- REQ-FRES-INFRA-F-M-11 (GitOps and infrastructure mechanisms): CumuCore subscription updates do not demonstrate changes to deployment descriptors or infrastructure redundancy through GitOps. That requirement needs a separate implementation path or an agreed scope decision.
- REQ-FRES-NET-F-M-12 (on-demand configuration): differential updates reduce unnecessary writes, but their disruption and activation time must be measured for an active session (Sections 5.3.4 and 6.3).
- REQ-FRES-NET-NF-M-13 (core performance monitoring): radio telemetry and core context are acquired. Monitoring coverage for AMF, UPF and SMF performance must be demonstrated explicitly (Sections 5.1.4 and 6.1).

2 RELATED WORK

This section reviews the main resilience mechanisms relevant to the SAFE-6G Resilience Trust Function. It first considers session continuity and network-function redundancy mechanisms defined for the 3GPP 5G system, followed by self-organising and self-healing network management and resilience mechanisms for virtualised and cloud-native infrastructure. It then examines user-centric and AI-driven approaches that introduce analytics and closed-loop control at finer levels of granularity.

Together, these mechanisms provide important building blocks for resilient mobile networks, but they do not by themselves close the per-session trust-control loop addressed by SAFE-6G. Section 2.6 therefore consolidates the gap identified across these areas and explains how the Resilience Trust Function complements the existing mechanisms.

2.1 RESILIENCE IN MOBILE NETWORKS: FROM NETWORK-WIDE REDUNDANCY TO PER-SESSION RECOVERY

Resilience includes the ability to maintain useful service during a disturbance and to recover afterwards. For this deliverable, three observations must be separated:

- **Supporting infrastructure:** whether the underlying network and core components are healthy and available.
- **UE session configuration:** whether the configuration accepted for a particular UE session meets the intended resilience requirements.
- **Service experienced by the UE:** whether the application or service actually maintains continuity and recovers after a disturbance.

A healthy core component does not establish that a particular application meets its continuity target, and a successful configuration write does not establish recovery. NFV resiliency requirements explicitly address service continuity as well as automated recovery and failure containment [5]. This motivates the session-level evaluation boundary used in SAFE-6G.

2.2 RESILIENCE MECHANISMS IN THE 3GPP 5G SYSTEM

3GPP TS 23.501 defines Session and Service Continuity (SSC) modes and network-function redundancy mechanisms [3]. The SSC mode selected for a PDU session remains associated with that session: SSC mode 1 preserves its anchor, while mode 3 supports establishing connectivity through a new anchor before releasing the previous connectivity. Subscription settings influence mode selection, but changing those settings is not equivalent to changing the mode of an existing PDU session. NF Sets support interchangeable function instances subject to their context and deployment arrangements. These capabilities provide underlying continuity mechanisms; their use in a SAFE-6G trust-control loop requires an explicit mapping from an approved action to the applicable core procedure.

2.3 SELF-ORGANIZING AND SELF-HEALING NETWORK MANAGEMENT

3GPP TS 32.541 describes self-healing within Self-Organizing Networks (SON), including fault detection, diagnosis and recovery [4]. Its management view complements the per-session control in this deliverable. A session-level controller can respond to the effects of a fault, while the management system repairs or compensates for its cause. The two loops need coordination to avoid competing changes during recovery; this document does not assume that the Resilience Trust Function replaces SON fault management.

2.4 RESILIENCE IN VIRTUALISED, CLOUD-NATIVE INFRASTRUCTURE

ETSI GS NFV-REL 001 addresses the resiliency of virtualised network functions and their supporting infrastructure, including automated recovery, redundancy, failure containment and service availability [5]. These mechanisms are relevant to the platform hosting SAFE-6G and to the core functions on which it depends. Their deployment is distinct from selecting a new QoS profile or UPF for a UE: both layers can contribute to service resilience, but each requires its own validation evidence.

2.5 USER-CENTRIC AND AI-DRIVEN RESILIENCE CONTROL

3GPP TS 23.288 specifies the Network Data Analytics Function (NWDAF), including analytics concerning network-function load, UE mobility and UE communication [10]. Analytics can target a UE or a group of UEs, so per-UE analytics are already part of the standardised toolbox. ETSI GS ZSM 009-1 provides services for governing closed loops, including their lifecycle and policy control [11]. These are relevant architectural precedents for separating observation, decision and execution. The particular SAFE-6G nLoTW, aLoTW and cLoTW signals, together with their reconciliation through CoCo, remain project-specific contracts.

2.6 GAP ADDRESSED BY THE SAFE-6G RESILIENCE TRUST FUNCTION

The mechanisms reviewed above provide resilience capabilities at different levels, including session continuity, network-function redundancy, self-healing, and VNF recovery. However, these mechanisms do not, by themselves, provide a per-session control loop that starts from an individual UE's resilience requirement and dynamically determines and applies the appropriate network configuration. In particular, they address continuity, redundancy, infrastructure recovery, and network management, but do not provide a unified mechanism that links a user-specific resilience requirement to a concrete and reportable reconfiguration of that UE session.

The SAFE-6G Resilience Trust Function addresses this gap by connecting resilience assessment, decision-making, and network enforcement at the session level. For a given UE session, it evaluates the achievable resilience trust level, translates the resulting resilience decision into a specific core-network configuration, and reports the outcome of the enforced configuration back to the Cognitive Coordinator. In this way, the function complements the existing standardised resilience mechanisms rather than replacing them.

3 TRUST FUNCTION DEFINITION & ROLE IN SAFE-6G

3.1 DEFINITION OF THE RESILIENCE TRUST FUNCTION

The User-centric Resilience Trust Function addresses the resilience dimension of trustworthiness: the capacity of a UE session and its associated core network configuration to withstand, adapt to, and recover from degraded conditions or connection faults. In practical terms, the function is designed to change core network configuration, recover a degraded UE connection, and increase tolerance to connection failures, all while continuously monitoring the UE and the surrounding network conditions.

The entities whose resilience is evaluated are individual UE sessions, identified by user and session identifiers and differentiated by device type such as smartphones and extended-reality, XR and devices. The Resilience Trust Function evaluates and enforces resilience at this per-session, user-centric granularity rather than applying a uniform, network-wide resilience policy.

3.2 TRUSTWORTHINESS SIGNALS: INPUTS AND OUTPUTS

The Resilience Trust Function receives three main categories of input for a UE session: the non-calibrated resilience trust level (nLoTW) provided by CoCo, network and UE telemetry gathered through continuous monitoring, and the calibrated resilience trust level (cLoTW), returned by CoCo after cross-function reconciliation, together with the corresponding start-or-stop instruction.

Based on these inputs, the function computes the achievable resilience trust level (aLoTW), which represents the resilience level that can practically be delivered under the current network conditions. It also derives the candidate actions associated with the session and reports them to CoCo. Once the calibrated result is received, the function applies the corresponding resilience flavour and, where required, performs the associated core-network configuration changes. Its outputs therefore include the computed aLoTW, the candidate actions proposed to CoCo, the resilience flavour applied to the session, and the outcome of the resulting configuration change.

3.3 RESILIENCE LEVELS, FLAVOURS, AND ENFORCEMENT POLICIES

The Resilience Trust Function distinguishes between the requested resilience level, the achievable resilience level, and the resilience flavour used for enforcement. The requested resilience level represents the resilience target initially provided through the trust-control workflow. The local-ai-agent evaluates the current network and UE context and computes the achievable resilience level (aLoTW) on a 0–100 scale, indicating the level that can practically be supported under the observed conditions. CoCo subsequently reconciles this assessment with the other trust dimensions and returns the calibrated resilience level (cLoTW).

This resolved value is then associated with one of the discrete resilience flavours (low, medium, or high) which selects the corresponding enforcement policy used by the vApp and nApp. The flavour therefore represents an enforcement policy category rather than a continuous resilience measurement or a probability of surviving a failure.

Two considerations justify the discretisation. First, the enforcement mechanism is itself discrete by construction: the vApp's Rules predictor is not one continuous model but six independent models, one per flavour and device type (Section 5.2.5), so a continuous resilience value would in any case have to be routed to one of a fixed set of discrete policies. Second, many of the parameters the nApp ultimately configures are themselves discrete or coarsely quantised 3GPP parameters (Section 5.3.3), so a bounded set of named flavours is what makes the resulting configuration reviewable as a whole, rather than as a parameter-by-parameter mapping from an arbitrary continuous input.

Three flavours are defined: low, medium and high. Section 5.2.5 identifies their device-specific models, and Section 5.3 describes the configuration fields they can affect. The numerical score-to-flavour boundaries and any hysteresis must be specified consistently in the implementation and its tests; this document does not infer those boundaries from the flavour names.

<i>Level of Resilience</i>	<i>Flavour</i>	<i>Enforcement intent</i>
Low	low	The vApp's Rules predictor selects the least constrained set of actions; minimal reconfiguration of QoS and user-plane parameters
Medium	medium	A balanced set of actions is selected, adjusting QoS flow configuration and user-plane function selection to a moderate degree
High	high	The most constrained set of actions is selected, maximising the adjustments applied to QoS flow configuration and user-plane function selection

Table 1: Level of Resilience levels, flavours and their enforcement intent

A high flavour should be understood as a policy intended to meet a more demanding resilience target. It does not necessarily require more writes, a higher numerical value for every QoS field, or migration to another UPF. If the current configuration already matches the policy, the appropriate outcome may be to retain it. Conversely, an action that protects one session can consume capacity or affect another session, so approval must consider the shared resource context.

The score-to-flavour mapping and the action model form separate stages. Their versions should be recorded together: changing a threshold without changing the model may alter which policy is selected for the same observation. Boundary tests should cover both sides of each transition, the exact boundary value and invalid inputs. Where measurements fluctuate, a documented dwell time or hysteresis can reduce repeated policy switching; its value needs to be chosen against the permitted response delay.

3.3 INTERACTION WITH THE COGNITIVE COORDINATOR

The Resilience Trust Function contributes to the overall LoTW framework by reporting its achievable resilience assessment to CoCo for reconciliation against the non-calibrated level requested and against the assessments of the other Trust Functions. CoCo returns the calibrated result, which the function then uses both to determine enforcement actions and, through the same message type, to start or

stop active monitoring of a session. Consistent with the agreed scope of this deliverable, the internal mechanisms by which CoCo itself computes or calibrates these values, and how it resolves conflicts among Trust Functions, are not elaborated here; this deliverable describes only the messages the Resilience Trust Function sends to and receives from CoCo. [1]

The function supports an initial session assessment and periodic evaluation of sessions under active monitoring. Enforcement translates a CoCo-approved decision into a core-network request and reports its processing outcome. Read-back of the effective configuration and observation of the resulting service quality are separate validation steps, discussed in Section 6.6.

The two-phase interaction gives CoCo an opportunity to reconcile candidate actions before the nApp changes the core. Its effectiveness depends on an approval remaining associated with the same proposal and a sufficiently recent session context. A delayed approval must not silently authorise a newer or unrelated candidate action set.

3.4 INTERACTION BOUNDARIES WITH THE OTHER TRUST FUNCTIONS

The Resilience Trust Function interacts with two components: CoCo, through asynchronous messaging carrying the resilience trust request, the achievable resilience report, and the resolved outcomes; and the core network, for reading UE and network state and for applying configuration changes. The Resilience Trust Function does not interact directly with the other Trust Functions (Safety, Security, Privacy, Reliability); any reconciliation between resilience and other trust dimensions is performed by CoCo, and is outside the scope of this deliverable, per the working boundary agreed for this document.

This is a deliberate architectural property rather than an incidental one: it means there is no interface surface between Trust Functions to specify, version or test, and that each function can be developed, deployed, restarted and reasoned about independently of the others. The cost is that the Resilience Trust Function cannot observe why its own achievable-resilience assessment was overridden — it receives only the calibrated value it is asked to enforce.

The function's responsibilities are bounded to the resilience dimension for a given UE session: assessing achievable resilience, applying the corresponding core network configuration changes, and monitoring and reporting outcomes. Explicitly outside its scope are the internal logic of CoCo; the internal logic of the other Trust Functions; the initial, one-time deployment of the function itself, performed through the edge-cloud continuum platform rather than by the function at runtime; and system-level validation and field-performance results, addressed in Work Package 5 (WP5). Component-level results are reported in Section 5.

4 REFERENCE ARCHITECTURE

4.1 ARCHITECTURAL OVERVIEW AND DESIGN PRINCIPLES

The reference architecture distributes responsibility across the local-ai-agent, vApp and nApp under CoCo coordination [1]. Runtime dependencies include OpenCAPIF for CMC API discovery and authorisation, CumuCore for context and configuration, Prometheus for radio telemetry, and MLOps platform for model training, deployment, and serving. In the current implementation description, CumuCore provides the core state-read and rule-execution functions; the architectural NEF role does not imply an additional, separately deployed NEF mediation path.

The architecture follows four main design principles:

1. **Separation of decision from enforcement.** The local-ai-agent obtains authorised access to core context and estimates achievable resilience, while the nApp is responsible for configuration writes. Access rights should reflect these separate responsibilities. The local-ai-agent therefore requires credentials or authorisation material for its permitted reads, even though it does not perform core reconfiguration.
2. **Per-device-type specialisation.** Because resilience and connectivity requirements are not uniform across device types, separate models are trained and deployed for smartphones and XR devices at both the local-ai-agent (Section 5.1.5) and the vApp (Section 5.2.5), rather than approximating both with a single generalist model.
3. **Bounded handling of partial failure.** The local-ai-agent has a deterministic fallback for model-serving failure, while the vApp can preserve the current configuration if action mapping fails. The nApp uses differential updates to avoid unnecessary writes. Differential updates alone do not provide transactional rollback, duplicate suppression or recovery from a partially completed multi-step change (Section 6.4).
4. **Best-effort handling of supporting operations.** Supporting operations are explicitly best-effort, while the core configuration write itself is not. The short-lived preemption record created by the nApp alongside a configuration change, and the outcome report sent to CoCo, are both best-effort and do not block or alter the primary configuration response if they fail (Sections 5.3.5 and 6.4).

The following figure illustrates this architecture, showing how the local-ai-agent, vApp, and nApp cooperate internally and interact with CoCo, DataOps, CumuCore, and the SAFE-6G continuum planes.

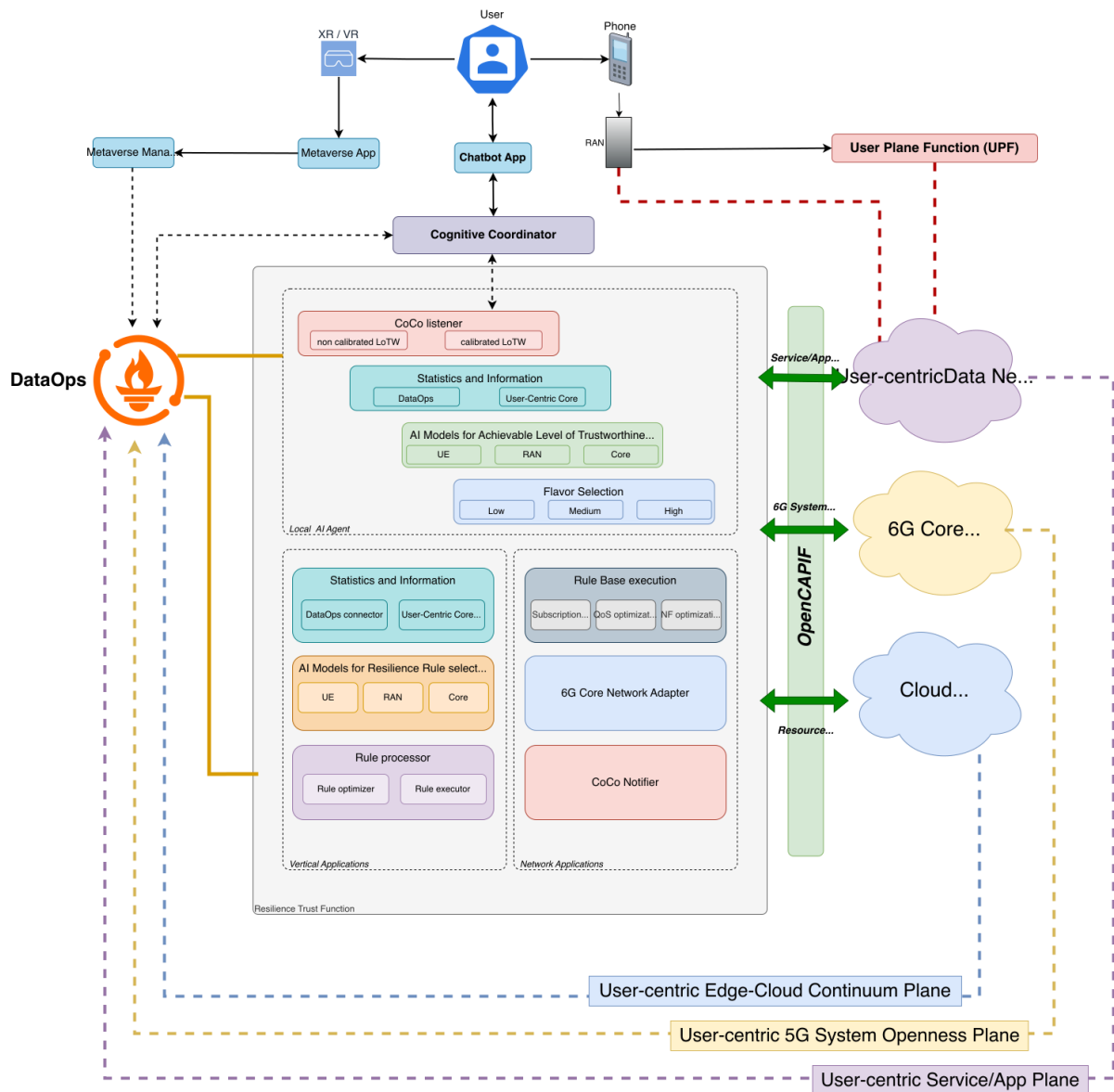


Figure 1: High-level architecture of the User-centric Resilience Trust Function

4.2 COMPONENT DECOMPOSITION

The local-ai-agent is responsible for determining, for a given UE session, the achievable resilience trust level and the applicable resilience flavour. On receiving a resilience trust request from CoCo, carrying the non-calibrated resilience trust level (nLoTW) for the session, it discovers and authorises access to the CumuCore API through OpenCAPIF, reads the session's UE, radio (gNB), user-plane (UPF), and slice context from the CumuCore API, and, where needed, the current UE or core state from the same API's 6G Core functions; in parallel, it queries Prometheus directly for the serving gNB's radio-network telemetry (Section 5.1.4). It then runs a LoTW predictor model hosted by MLOps and reports the resulting achievable score back to CoCo. Once CoCo returns a resolved, calibrated resilience trust level (cLoTW) for the session together with a start or stop instruction, the local-ai-agent instructs the vApp to add or remove the corresponding user from active monitoring.

The vApp is responsible for the session-level orchestration of resilience once a UE has been placed under active monitoring. Periodically, for each actively monitored user, it discovers and authorises the CumuCore API, reads the session's UE, radio (gNB), user-plane (UPF), and slice context from it, and runs a distinct Rules predictor model hosted by MLOps to determine the applicable actions. It reports this outcome, together with the candidate list of actions, back to CoCo. When CoCo grants permission and returns a final resolved score for the session, the vApp correlates the response by session identifier, reuses the pending actions it had proposed, and requests the nApp to apply the corresponding UE configuration.

The nApp is responsible for enforcing the resolved resilience actions on the core network. It discovers and authorises the CumuCore API through OpenCAPIF and executes the required UE subscription, session, and slice or user-plane function updates through this API, which realises the 6G Core rule-execution and state-read functions referenced in the reference architecture; no separate NEF-mediated system is used in the current implementation. It then reports the outcome of the configuration change back to CoCo.

4.3 MESSAGE-DRIVEN INTERACTION MODEL

The Resilience Trust Function uses asynchronous messaging for its interaction with CoCo and Agent-to-Agent (A2A) requests for communication between its internal components. The main interactions are summarised in the following table, identifying the sender, receiver, message or task, and its purpose.

The local-ai-agent receives the initial resilience trust request from CoCo, reports the computed achievable resilience level, and later receives the resolved score used to start or stop monitoring. Once monitoring is active, the vApp periodically evaluates the session, reports candidate outcomes to CoCo, and, after receiving approval, requests the nApp to apply the selected configuration. The nApp performs the corresponding core-network update and reports the resulting configuration outcome back to CoCo.

Interaction with CoCo takes place entirely through asynchronous messaging, using six distinct message types: a resilience trust request and an achievable-resilience report exchanged with the local-ai-agent; a resolved score, used by CoCo both to start and to stop monitoring for a session, also exchanged with the local-ai-agent; an intermediate outcome and a final resolved score exchanged with the vApp; and a final configuration outcome reported by the nApp.

The architecture currently has no separate message type for confirming that a requested flavour has been activated; this feedback is instead represented through the achievable-resilience report and the intermediate outcome messages. When CoCo does not grant permission for a proposed action, it simply withholds the final resolved score, and the vApp does not proceed to request any configuration change from the nApp; no explicit rejection message is defined.

The Resilience Trust Function does not interact directly with the other Trust Functions; any such interaction is mediated by CoCo and falls outside the scope of this deliverable.

4.4 TOPIC CATALOGUE AND MESSAGE SCHEMAS

The table below summarises the CoCo topics and inter-layer tasks, including their direction and purpose. The field-level contracts are described in Sections 5.1.3, 5.1.6, 5.2.3, 5.2.6, 5.3.3 and 5.3.5. The topic names are implementation identifiers [7]–[9].

Topic/Task	Mechanism	Producer -> Consumer	Purpose
resilience_trust_score	Kafka	CoCo → local-ai-agent	Opens session assessment (nLoTW)
resilience_report	Kafka	local-ai-agent → CoCo	Achievable resilience report (aLoTW)
resilience_resolved_score	Kafka	CoCo → local-ai-agent	Calibrated score; start or stop monitoring (cLoTW)
resilience_outcome	Kafka	vApp → CoCo	Intermediate outcome each periodic cycle
resilience_resolved_final_score	Kafka	CoCo → vApp	Final resolved score; only when permission is granted
resilience_final_outcome	Kafka	nApp → CoCo	Configuration result, best-effort
add_user / remove_user	A2A	local-ai-agent → vApp	Start/stop monitoring
update_UE_configuration / read_UE_configuration	A2A	vApp → nApp	Apply or read UE configuration

Table 2: Message Contracts

The trust-control loop crosses several security boundaries. Kafka topics carry session identifiers, trust levels and candidate actions; A2A requests trigger monitoring and configuration operations; OpenCAPIF governs discovery and authorisation; and CMC accepts state-changing requests. The integration design should therefore define authentication for every producer and consumer, least-privilege scopes for read and write operations, transport protection, credential rotation, topic-level access control and audit logging.

Sensitive subscriber identifiers should be minimised in logs and retained only for the period required for operation and validation. Message handling should also include controls against unauthorised or replayed requests, particularly where messages or A2A tasks can result in configuration changes. Security testing should include expired credentials, forged or duplicated messages, unauthorised topic access, invalid identifiers and attempts to apply actions outside the permitted subscriber or slice scope.

4.5 DEPLOYMENT VIEW

The three components operate as long-lived services within the Kubernetes deployment rather than as per-session workloads. Each component exposes a dedicated health-check endpoint, while the vApp additionally maintains the set of UE sessions under active monitoring. The components therefore retain application-level responsibilities independently of the Kubernetes lifecycle: Kubernetes provides the execution and restart environment, whereas session assessment, monitoring, decision-making, and enforcement remain responsibilities of the application components themselves.

Failure handling is implemented at component level for selected conditions. The local-ai-agent applies a deterministic fallback when the MLOps prediction service cannot be reached, while the vApp includes retry-and-backoff handling for transient failures in its outbound requests and can preserve the current configuration when action mapping cannot be completed. At the nApp, supporting operations such as the preemption record and outcome reporting to CoCo are best-effort and do not block the primary configuration response. Behaviour when telemetry retrieval from CumuCore or Prometheus fails remains an integration item and is not currently defined for all components.

The default deployment uses a single replica for each component, with horizontal pod autoscaling defined but disabled. Consequently, the current deployment does not establish coordinated ownership of monitored sessions across multiple vApp replicas. If horizontal scaling is enabled, session state, ownership, reassignment after replica failure, and prevention of duplicate evaluations must be explicitly addressed. Kubernetes can create and restart replicas, but it does not by itself provide these application-level session semantics. Persistence of the vApp monitoring state and its recovery after restart therefore remain deployment and integration validation items.

4.6 END-TO-END INTERACTION

The end-to-end lifecycle begins when CoCo sends a resilience trust request for a UE session to the local-ai-agent. The local-ai-agent gathers the relevant UE, radio, UPF, and slice context, invokes the LoTW predictor, and reports the resulting achievable resilience level (aLoTW) back to CoCo. It then reconciles this assessment and, when appropriate, returns a resolved score together with a start instruction. The local-ai-agent uses this response to register the UE session with the vApp for active monitoring. Once monitoring is active, the vApp periodically acquires updated context, evaluates the session using the corresponding Rules predictor, and reports the resulting achievable score and candidate actions to CoCo.

The main decision point occurs when CoCo evaluates the proposed actions. If CoCo grants permission, it returns a final resolved score for the corresponding session and proposal. The vApp then forwards the approved configuration to the nApp, which applies the required changes through the CumuCore API and reports the execution outcome. If no final resolved score is received, the vApp does not request any configuration change from the nApp for that monitoring cycle. The monitoring cycle continues for as long as the session remains active. When CoCo issues a stop instruction, the local-ai-agent requests the vApp to remove the UE from active monitoring, terminating the resilience-control loop for that session. The stop branch is part of the defined message contract, although its emission by the live CoCo integration remains an item to be validated.

Figure 2 summarises this lifecycle, from initial assessment and activation to periodic evaluation, approval, enforcement, and session termination. Confirmation that the applied changes actually improve the UE's experienced service requires the additional post-action verification described in Section 6.6.

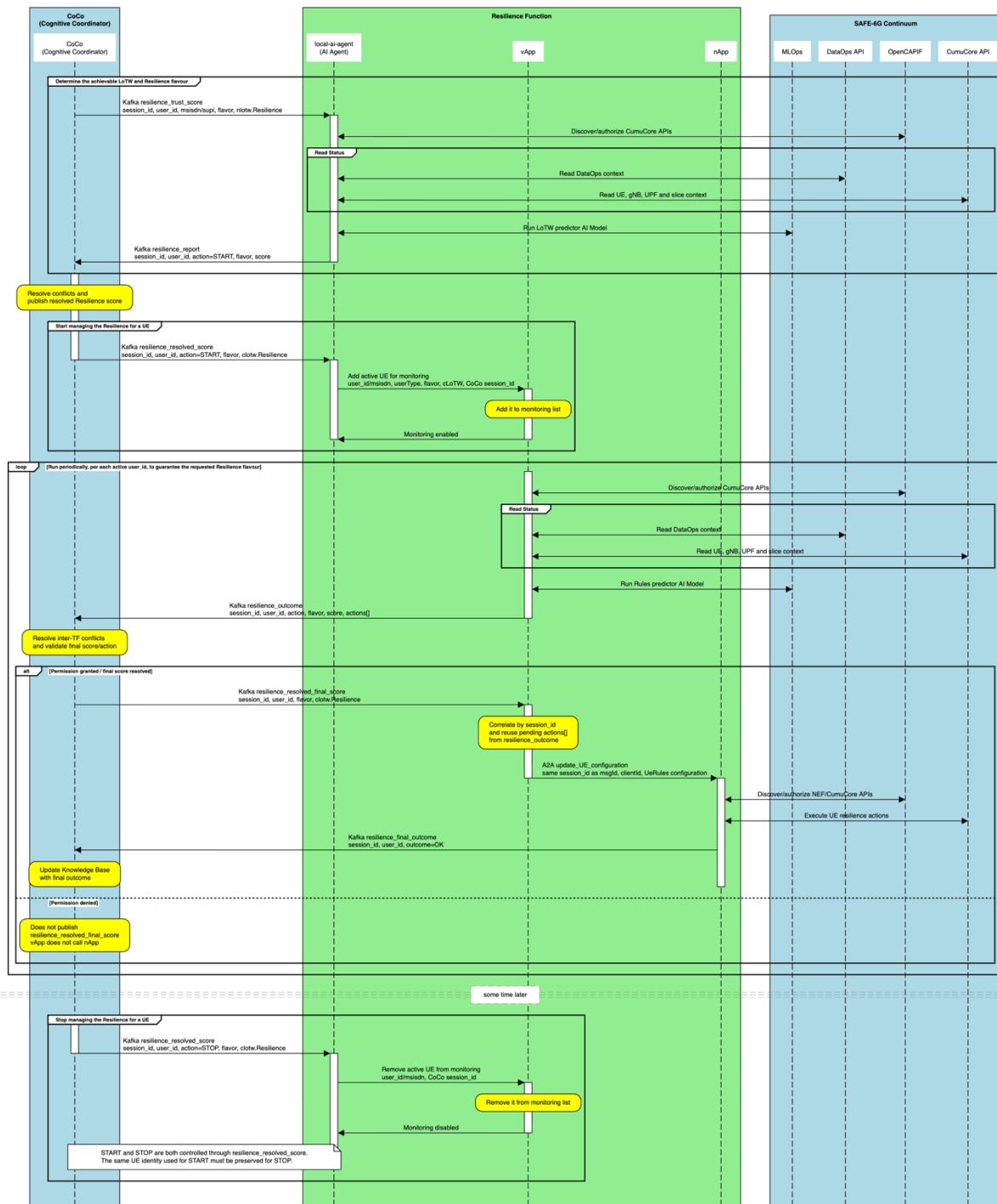


Figure 2: Resilience Trust Function session lifecycle

4.7 ARCHITECTURAL ASSUMPTIONS AND CONSTRAINTS

The architecture rests on the following assumptions, and they bound every claim made in the remainder of this deliverable.

1. **CoCo protocol.** The function assumes a Cognitive Coordinator that implements both phases of the contract described in Section 3.4: an initial trust request followed, later, by a resolved score. Whether a session for which the resolved score never arrives is eventually released or retained indefinitely has not been confirmed in the material reviewed.
2. **Message broker.** The function assumes a Kafka-based message broker is available for communication with CoCo. Specific delivery guarantees — ordering across topics, exactly-once delivery, or replay of messages published while a component was not running — have not been characterised in the material reviewed and should not be assumed either way.
3. **Agent-to-agent mechanism.** The function assumes an agent-to-agent (A2A) request mechanism is available between the local-ai-agent and the vApp, and between the vApp and the nApp.
4. **OpenCAPIF.** The function assumes OpenCAPIF is available as the mechanism for discovering and authorising access to the CumuCore API. Unavailability of OpenCAPIF itself, as distinct from CumuCore, has no described fallback (Section 7).
5. **Prometheus.** The function assumes Prometheus is available for gNB radio-network telemetry.
6. **CumuCore object model.** The function assumes CumuCore exposes the object model it depends on: UE, radio, user-plane and slice context that can be read, and subscription profile, slice and application-session-with-QoS records that can be read, created and updated (Section 5.3.4).
7. **Pre-existing subscriber.** The function assumes the UE session's subscriber already exists on the core; the nApp modifies an existing subscription profile and slice or user-plane assignment rather than creating a new subscriber record.
8. **Single-writer assumption.** The function assumes it is the only writer of the subscription profile it modifies for a given UE. This is not guaranteed by the architecture: the profile the nApp updates in place could, in principle, be modified concurrently by another process or operator action, a possibility not addressed in the material reviewed.

An architectural constraint of the current design is that the UE identity used to start monitoring a session must be preserved through to the corresponding stop instruction, since both are controlled through the same resolved-score message type.

CoCo, OpenCAPIF, CumuCore, Prometheus and MLOps are runtime dependencies of the function. The edge-cloud continuum handles initial deployment and continues to provide the compute, networking and service hosting required during operation. Deployment orchestration is outside the function's own control loop, but loss of its hosting infrastructure can still interrupt that loop.

4.8 SESSION STATE AND MESSAGE CORRELATION

The asynchronous workflow requires a distinction between a session identity and an individual decision attempt. A session can undergo many monitoring cycles, and several proposals may exist over its lifetime. A session identifier alone is therefore insufficient to prove that a late approval refers to the action currently held by the vApp. The following logical states provide a proposed basis for integration tests; they do not add new implemented Kafka topics.

State	Entry condition	Permitted next step
Assessing	Initial trust request accepted	Collect context, estimate aLoTW and publish the assessment.
Awaiting activation	Assessment published	Start monitoring after a matching resolved start instruction.
Monitoring	Session registered in the vApp	Acquire a fresh snapshot and prepare a candidate.
Awaiting approval	Candidate sent to CoCo	Wait for approval; expire the candidate if its validity ends.
Applying	Approval matches a valid candidate	Request the nApp update and capture the execution outcome.
Reconciling	Execution finished or its outcome is uncertain	Read effective state and decide whether to resume or escalate.
Stopped	Matching stop instruction processed	Cancel pending work and suppress later execution for that session.

Table 3: Logical states for resilience session validation

The message envelope should identify the session, subscriber and individual proposal, with creation and expiry times and a schema version. These are recommended extensions where equivalent fields are absent. Each proposal identifier should refer to one candidate action set. Consumers should reject expired approvals and approvals received after stop, retaining the rejection reason.

Kafka transactions alone do not ensure exactly-once updates to an external system such as CumuCore [23]. Tests should cover duplicates, timeout retries, consumer restarts and out-of-order responses. Repeated updates should converge on the intended state; application-session creation needs protection against duplicate resources.

5 TRUST FUNCTION LAYERS

5.1 LOCAL AI AGENT

The local-ai-agent is the entry point of the Resilience Trust Function for the initial assessment of a UE session. This section describes its role in the resilience workflow, its implementation stack and message interfaces, the telemetry and preprocessing used to construct the model inputs, the decision-making process used to estimate the achievable resilience level (aLoTW), and the outputs, limitations, and failure-handling mechanisms associated with this component.

5.1.1 ROLE AND POSITION IN THE RESILIENCE WORKFLOW

Within the Resilience Trust Function, the local AI agent is the first component activated for a UE session. Its main responsibilities are:

- **Session initiation:** determining the achievable resilience trust level for a UE session when resilience handling starts.
- **Handover to the vApp:** handing the session over to the vApp for ongoing monitoring once CoCo has confirmed the resilience handling.
- **Session termination:** instructing the vApp to stop monitoring the session when CoCo signals that resilience handling should end.
- **Periodic re-evaluation:** this is not performed by the local AI agent; the periodic re-evaluation of active sessions is handled by the vApp (Section 5.2).

5.1.2 IMPLEMENTATION STACK

The local-ai-agent is implemented as a containerised service. It communicates with CoCo over an asynchronous message broker, discovers and authorises access to the CumuCore API through OpenCAPIF, and calls the MLOps-hosted LoTW predictor and the Prometheus query API over HTTP. [7]

5.1.3 INPUTS AND MESSAGE CONTRACT

The local AI agent interacts with CoCo through two message types during the session lifecycle:

- **Resilience trust request:** this message triggers the local AI agent and carries the session identifier, user identifier, subscriber identity (MSISDN or SUPI), requested resilience flavour, and the non-calibrated resilience trust level for the resilience dimension (nLoTW).
- **Resolved score:** this message is received later in the session lifecycle and carries the session and user identifiers, an action (start or stop), the resilience flavour, and the calibrated resilience trust level (cLoTW). The same message type is used both to start and stop session monitoring, with the action field indicating the requested operation.

5.1.4 PREPROCESSING AND TELEMETRY

On receiving the initial trust request, the local-ai-agent discovers and authorises access to the CumuCore API through the OpenCAPIF SDK, then retrieves the session's UE, radio (gNB), user-plane (UPF), and slice context, and, where needed, current UE or core state, all from the CumuCore API.

In parallel, the local-ai-agent queries Prometheus for the serving gNB's configured radio metrics: downlink and uplink PRB utilisation, RRC establishment attempts and successes, initial UE context setup attempts and successes, context drops, and handover attempts and successes. Handover measurements distinguish the configured inter-gNB and inter-CU cases. Event counts, utilisation gauges and precomputed rates need separate handling; they should not all be described as raw counters. Success percentages are derived using compatible measurement scopes and intervals [15].

This radio telemetry is complemented by capacity data: the gNB's serving frequency band, mapped to a coarse frequency category, and the maximum number of UEs the gNB can register and keep connected, both drawn from a local, per-gNB configuration table maintained by the local-ai-agent rather than queried live; the currently registered and connected UE counts used to compute occupancy ratios against these maxima are read from CumuCore. Together, these form the feature set used in the decision-making step described next.

The model feature contract should state the name, unit, source, entity scope and missing-value treatment of each input. The serving gNB, candidate neighbouring gNBs and UPFs must remain identifiable throughout preprocessing. Reordering two numeric fields can produce a syntactically valid request with the wrong meaning, so feature order and schema version are part of the model interface.

Cell-level measurements are shared context, not direct measurements of every UE attached to that cell. For example, a successful aggregate handover rate can coexist with poor behaviour for one device or application. The session assessment should therefore preserve the distinction between per-cell observations, per-UPF load and user-specific configuration. This also helps explain which inputs changed when two sessions receive different scores.

5.1.5 DECISION-MAKING APPROACH

For decision-making, a MLOps layer has been integrated which incorporates a set of machine learning models developed to assess network resilience.

5.1.5.1 MODULE OBJECTIVE

This module estimates aLoTW, the Achievable Level of Trustworthiness, on a 0–100 scale. Its models combine radio, UPF and device context to approximate the resilience score assigned by the training policy. Because training labels originate in a manually defined knowledge seed, the output is a policy-derived estimate. It does not by itself guarantee a service level or quantify a probability of uninterrupted service.

The prediction informs the subsequent selection of a resilience policy. The currently described enforcement catalogue covers QoS, subscription and slice or UPF changes; a direct RAN handover command is not included in that catalogue.

5.1.5.2 ALGORITHM SELECTION AND MODEL VARIANTS

Random Forest regression is used for score prediction. The method combines decision trees and can represent nonlinear relationships between tabular features [17]. Its suitability for the reported deployment depends on the training distribution, model size and serving resources; using synthetic training data does not itself establish robustness. CPU inference avoids a mandatory GPU dependency, but latency must still be measured for the actual serving workload.

Separate model instances are trained for smartphones and XR glasses. Both use the same algorithm family, with device-specific labels or datasets allowing different policy responses. This design choice needs comparison with a shared-model baseline before a benefit in predictive accuracy can be claimed.

Therefore, two independent models were developed:

- **rf_p model:** Designed and optimized for smartphones.
- **rf_g model:** Tailored for augmented reality glasses.

5.1.5.3 WORKFLOW, SYNTHETIC DATA GENERATION, AND MODEL EVALUATION

Since real network data is unavailable for model training, synthetic datasets were generated for this purpose. The different modules of the SAFE-6G MLOps pipeline were used to support the model training workflow, including dataset preparation, training configuration, model training, and subsequent deployment for inference. The overall workflow is shown in the following diagram.

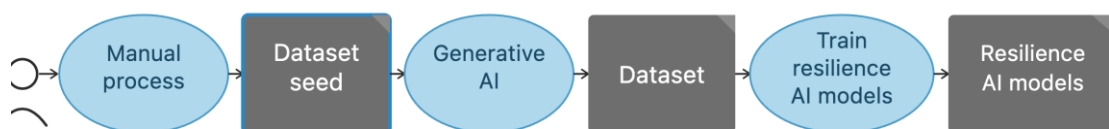


Figure 3: MLOps workflow for synthetic data generation, model training, and deployment

The workflow is as follows:

1. A manually defined knowledge seed maps combinations of categorical network conditions to target aLoTW labels. These labels express the chosen policy assumptions and serve as the supervised targets for training; they are not measurements of real-world resilience.
2. The categorical combinations are expanded into numerical samples within the prescribed ranges, producing datasets of 300,000 rows. The generated data inherit the combinations and sampling assumptions encoded by the generator.
3. Once these synthetic datasets were available, hyperparameters were configured, and model training proceeded.

The following table summarizes the possible combinations defined in the knowledge seed (step 1), which is used to generate the synthetic dataset.

<i>Variable</i>	<i>Possible values</i>
frequency_band	low band (700MHz), mid band (3500MHz)
occupancy (connected gNB)	good, high — cannot be null as it always exists
low_occupancy (low-band neighbouring gNB)	good, high, null
mid_occupancy (mid-band neighbouring gNB)	good, high, null
upf_load_anchored	good, high (cannot be null)
upf_load_alternative	good, high (cannot be null)

Table 4: Variables of the Knowledge Seed

The categorical combinations obtained from Step 1 are expanded to generate a dataset containing 300000 records. During this process, each combination is replicated as many times as required, and its categorical variables are converted into numerical values within the predefined range associated with each category. As a result, the final dataset preserves the distribution of the categorical combinations while providing numerical values that are consistent with the defined ranges for each variable.

These ranges are defined in the following tables:

▪ Frequency Band

<i>Categorical value</i>	<i>Numeric range (MHz)</i>
Low band (700MHz)	< 1000 MHz
Mid band (3500MHz)	1000 – 6000 MHz

Table 5: Frequency Band Ranges

▪ Occupancy

<i>KPI</i>	<i>Range</i>	<i>good</i>	<i>high</i>
PRB DL	0–100%	< 80%	≥ 80%
PRB UL	0–100%	< 80%	≥ 80%
RRC Setup Success	0–100%	> 95%	≤ 95%
Retainability	0–100%	> 98%	≤ 98%
HO Success	0–100%	> 95%	≤ 95%
Accessibility Control Plane UE Initial Context Setup Success Rate	0–100%	> 95%	≤ 95%
Registered UE Ratio	0–100%	< 80%	≥ 80%
Connected UE Ratio	0–100%	< 80%	≥ 80%

Table 6: Occupancy Ranges

UPF Load

<i>KPI</i>	<i>Range</i>	<i>good</i>	<i>high</i>
UPF Data Plane Load	0–100%	< 80%	≥ 80%
UPF Signalling Plane Load	0–100%	< 80%	≥ 80%
UPF Computing Load	0–100%	< 80%	≥ 80%

Table 7: UPF Load Ranges

Once the datasets are defined, it's necessary to configure the hyperparameters of the models which control the learning process and the complexity of the models. Both models share the same hyperparameter configuration, determined through an iterative testing process aimed at balancing generalization capacity and computational cost. The established configuration is summarized in the following hyperparameters:

- **n_estimators = 300**: Number of trees in the forest.
- **max_depth = 30**: Maximum depth of each decision tree.
- **min_samples_split = 5**: Minimum number of samples required to split an internal node.
- **max_features = 0.7**: 70% of the available features are considered at each split.
- **random_state = 42**: Fixed seed to ensure reproducibility.
- **n_jobs = -1**: Uses all available CPU cores for parallel training.

The local AI agent accesses the prediction models synchronously by sending HTTP POST requests to the API exposed by KServe. Both models are deployed under the following base URL: [https://model.develop.resilience.function.\[Safe6G_domain\]/v1](https://model.develop.resilience.function.[Safe6G_domain]/v1)

Each model is exposed through a dedicated prediction endpoint corresponding to the target device type, allowing the local AI agent to invoke the appropriate model depending on whether the request is associated with a smartphone or XR glasses.

<i>Endpoint</i>	<i>Model</i>
POST /models/rf_p:predict	Random Forest – Smartphone
POST /models/rf_g:predict	Random Forest – XR Glasses

Table 8: ALoTW Endpoints

After building the models, a series of tests were conducted to demonstrate their efficiency. These tests are summarized in the following table:

<i>Category</i>	<i>Result (passed/failed)</i>
Infrastructure (registration, endpoints, payload validation)	9 / 0
Response schema (predicted_alotw, range [0,100])	6 / 0
Behavior (determinism, batching, latency, concurrency)	8 / 0

Error handling (invalid, null, out-of-range inputs)	202 / 0
Accuracy over the 144 seed combinations (x5 arrays)	1437 / 3

Table 9: Results of ALoTW Models Testing

The four interface and behaviour categories report 225 passes and no failures. The seed-agreement category reports 1,437 passes and 3 failures out of 1,440 checks, equivalent to 99.79% passing. Its stated tolerance is 5%; the test specification must distinguish a relative percentage from five points on the 0–100 score scale. This pass rate is not classification accuracy and does not establish performance on real network observations.

A series of tests were also run to evaluate model errors. The selected evaluation metrics are:

- **RMSE:** Mean error penalizing large deviations (lower is better).
- **MAE:** Direct average absolute error (lower is better).
- **R²:** Model precision percentage (closer to 1 is perfect).

<i>Device</i>	<i>RMSE</i>	<i>MAE</i>	<i>R2 Score</i>
Phone	0.720	0.229	0.9995
Glasses	0.588	0.155	0.9997

Table 10: ALoTW Models Evaluation

Both device models have R² values above 0.999 on the reported synthetic evaluation. The XR model has lower RMSE and MAE on its own evaluation set. These results show close agreement with the generated targets under the tested distribution; they do not establish robustness to unseen field conditions or a statistically significant advantage between device models.

5.1.5.4 MODEL DEPLOYMENT AND INFERENCE INFRASTRUCTURE

The MLOps model serving module is used to expose the trained prediction models through an HTTP inference interface, with KServe acting as the underlying serving framework. The exposed endpoints follow the KServe V1 inference protocol [22]. Model artefacts are loaded as part of the serving deployment, and each inference request is routed to the model corresponding to the target device type before payload validation and prediction are performed.

Each time a prediction is requested, the execution flow is as follows:

1. **Model selection:** routes the request to the model associated with the device type.
2. **Preprocessing:** validates feature structure and ranges. For optional alternative-gNB features, the reported implementation uses -1 as a missing-value sentinel. That encoding must also be represented consistently during training; -1 is an explicit numeric convention, not an automatic guarantee of correct missing-data handling. Invalid required inputs result in HTTP 400.
3. **Predict Phase:** Performs the aLoTW prediction.

5.1.5.5 INTERPRETATION AND VALIDATION OF SYNTHETIC PREDICTIONS

Expanding the knowledge seed to 300,000 rows increases the number of numerical examples, but it does not add independent observations of network behaviour. The model learns the score policy embedded in the seed and its sampling ranges. This can be useful for testing a serving pipeline and interpolating within the chosen policy, provided that agreement with the policy is kept distinct from validation of the policy itself.

A useful evaluation design should separate at least two questions. First, does the regressor reproduce the intended seed policy for unseen numerical samples? Second, do the policy scores correspond to desirable session outcomes under actual disturbances? The first can be assessed with synthetic holdouts and boundary cases. The second needs observations of the network and application before and after an action, with an independent definition of acceptable service.

Data splitting should prevent samples derived from the same scenario template or collection period from leaking into both training and evaluation in a way that overstates generalisation. In addition to an ordinary numerical holdout, validation should reserve complete scenario combinations or operating ranges and report errors separately for each device type. A deliberately withheld scenario tests a different capability from a new random sample inside an already represented range.

Evaluation should also inspect the direction of the error. Overestimating achievable resilience can lead CoCo to approve a target that the network cannot sustain, whereas underestimation may cause unnecessary configuration changes or rejection of a feasible target. Reporting the frequency and magnitude of these two errors, together with MAE and RMSE, connects model evaluation to the operational decision. The acceptable limits must be selected by the project validation process.

A reproducible model release should bind the seed, generator configuration, feature schema, preprocessing code and model artefact to a versioned evaluation report. Real-data validation should then compare observed inputs with the training ranges and track prediction error where an independent target is available. Until such evidence exists, a high synthetic R^2 supports implementation consistency but not a field-performance guarantee.

5.1.6 OUTPUTS AND MESSAGE CONTRACT

Following the initial assessment, the local-ai-agent reports the achievable resilience value (aLoTW) to CoCo as a resilience report, carrying the session and user identifiers, a start action, the resilience flavour, and the computed score.

Once CoCo returns the resolved score with a start action, the local-ai-agent requests the vApp to add the corresponding user to active monitoring, passing the user identifier or MSISDN, the user type, the flavour, the calibrated resilience level, and the CoCo session identifier; the vApp acknowledges that monitoring has been enabled. When CoCo instead returns the resolved score with a stop action, the local-ai-agent requests the vApp to remove the user from active monitoring, passing the user identifier or MSISDN; the vApp acknowledges that monitoring has been disabled.

5.1.7 LIMITATIONS

As the design and training of the LoTW predictor model are addressed outside this deliverable, no assessment of its accuracy, data requirements, or explainability is provided here. Although the resolved-score message schema supports both a start and a stop action, only the start action is confirmed, in the current implementation, as a message CoCo actually issues; stop support is a forward-compatible allowance in the message contract, not yet confirmed as exercised by a live CoCo integration. Finally, the local-ai-agent's behaviour when context retrieval from Prometheus or CumuCore is incomplete or fails has not been confirmed and should be verified before being described with certainty.

5.1.8 FAILURE HANDLING

The local-ai-agent defines a confirmed fallback for one specific failure condition: if the model-serving call to the LoTW predictor cannot be completed, a deterministic rule adjusts the requested resilience level by a fixed, device-type-dependent amount, bounded to a valid range, rather than leaving the session unassessed. The current design has no separate message type for confirming that a requested flavour has been activated; this feedback is instead inferred from the resilience report and, later, from the vApp's intermediate outcome (Section 5.2).

5.2 VERTICAL APPLICATION (vAPP)

The vApp is the orchestration layer of the Resilience Trust Function for UE sessions that have already been placed under active monitoring. Unlike the local-ai-agent, which performs the initial achievable-resilience assessment and initiates or terminates monitoring, the vApp is responsible for the periodic re-evaluation of active sessions, the generation of candidate resilience actions, their submission to CoCo for approval, and the subsequent coordination of enforcement through the nApp. This section describes its role, implementation stack, message interfaces, telemetry and preprocessing, decision-making approach, outputs, limitations, and failure-handling behaviour.

5.2.1 ROLE AND POSITION IN THE RESILIENCE WORKFLOW

The vApp is responsible for orchestrating resilience handling for a UE session once the local-ai-agent has placed it under active monitoring. It maintains the set of actively monitored sessions, periodically re-evaluates each of them against the requested resilience flavour, proposes candidate actions to CoCo, and, once CoCo grants permission and returns a final resolved score, translates the associated actions into a concrete UE configuration request for the nApp. It also acknowledges the local-ai-agent's requests to start or stop monitoring a given session.

5.2.2 IMPLEMENTATION STACK

The vApp is implemented as a containerised service. It communicates with CoCo over an asynchronous message broker, exchanges Agent-to-Agent (A2A) protocol tasks with the local-ai-agent and the nApp, and discovers and authorises access to the CumuCore API through OpenCAPIF. A small internal retry-and-backoff mechanism makes the vApp's own outbound calls more robust to transient failures; this

is a general software-reliability mechanism internal to the service and should not be confused with the resilience trust dimension the function evaluates for UE sessions. [8]

5.2.3 INPUTS AND MESSAGE CONTRACT

The vApp receives add-user and remove-user tasks from the local-ai-agent. The add task carries the user identifier, device type, flavour, calibrated trust target and CoCo session identifier. A remove task identifies the monitored user; the session identifier must be retained wherever the deployed contract supports it. The final resolved score from CoCo authorises a pending proposal. An absent response does not distinguish rejection from delay or message loss (Sections 4.8 and 5.2.8).

5.2.4 PREPROCESSING AND TELEMETRY

For each monitored session, the vApp reads the UE, gNB, UPF and slice context and the current configuration from CumuCore after OpenCAPIF discovery and authorisation. It also acquires the radio metrics described in Section 5.1.4 and derives the required indicators. Their sampling time, scope and preprocessing should be consistent with the inputs expected by the selected model.

The implementation description also includes a static frequency-to-flavour lookup. Such a lookup is a project policy heuristic: frequency band alone is not evidence of greater resilience. Its influence should be documented and tested alongside congestion, radio quality and device requirements to avoid a contradictory mapping between the heuristic and the learned policy.

5.2.5 DECISION-MAKING APPROACH

The vApp uses MLOps-hosted action models to select a candidate configuration for the requested resilience flavour and the observed network state.

5.2.5.1 MODULE OBJECTIVE

This module produces eight action parameters from a 38-feature input vector, covering QoS-related configuration and UPF selection. The resulting candidate is sent to CoCo for reconciliation and approval. The vApp then maps the approved proposal into a request to the nApp, which performs enforcement. A model output is therefore a configuration proposal, not proof that the core can apply it or that it is optimal for the live session.

5.2.5.2 ALGORITHM SELECTION AND MODEL VARIANTS

The proposed solution aims to determine the most appropriate actions according to the current network conditions. These conditions are represented as categorical states that define the decision context. Therefore, although the input data are initially numerical, they are transformed into their corresponding categorical representation before being processed by the model. The network's current status is the same as shown in the local AI agent section.

The numerical observations are categorised into a finite state representation for tabular action selection. Tabular Q-learning is the reported training method [20]. With the configured discount factor

$\gamma = 0.0$, the update uses immediate reward only; future-state value contributes nothing. The resulting policy should therefore be described as maximising the configured immediate reward, rather than optimising long-horizon recovery.

The reported exploration policy uses an Upper Confidence Bound (UCB) term to balance estimated reward with exploration of less-sampled actions. UCB methods have theoretical results under specified bandit assumptions [21]; those results do not automatically transfer to every network-control environment. The discrete representation allows inexpensive action lookup during inference, subject to the actual table size and model-serving overhead.

The development account reports that the tested epsilon-greedy schedule decayed around episode 916 in a 50,000-episode run, motivating the use of UCB. This observation depends on that schedule and test environment. UCB encourages exploration but does not, by itself, guarantee that every state is visited or that a finite training run covers every state-action pair.

Since configuration targets are not uniform across device profiles or service scenarios, independent models were developed per flavour (high, medium, or low) and device type (smartphone or XR glasses). Consequently, six independent models were created:

- **rb_rl_high_phone**: High flavour and phone.
- **rb_rl_high_xr**: High flavour and xr glasses.
- **rb_rl_medium_phone**: Medium flavour and phone.
- **rb_rl_medium_xr**: Medium flavour and xr glasses.
- **rb_rl_low_phone**: Low flavour and phone.
- **rb_rl_low_xr**: Low flavour and xr glasses.

The hyperparameters were configured using the same settings, which are as follows:

- **n_episodes = 50,000**: Number of training episodes.
- **alpha = 0.1**: Learning rate used to update Q-values.
- **gamma = 0.0**: Discount factor; only immediate rewards are considered.
- **c = 2.0**: UCB exploration constant controlling the exploration–exploitation trade-off.
- **random_state = 42**: Fixed seed to ensure reproducibility.

The six Rule-Based Reinforcement Learning models are deployed under the same base URL as the aLoTW model: [https://model.develop.resilience.function.\[Safe6G_domain\]/v1](https://model.develop.resilience.function.[Safe6G_domain]/v1)

Each model is exposed through an independent prediction endpoint. This separation simplifies model selection and enables clients to directly invoke the appropriate model according to the required scenario.

<i>Endpoint</i>	<i>Model</i>
POST /models/rb_rl_high_phone:predict	High Resilience – Smartphone
POST /models/rb_rl_high_xr:predict	High Resilience – XR Glasses

POST /models/rb_rl_medium_phone:predict	Medium Resilience – Smartphone
POST /models/rb_rl_medium_xr:predict	Medium Resilience – XR Glasses
POST /models/rb_rl_low_phone:predict	Low Resilience – Smartphone
POST /models/rb_rl_low_xr:predict	Low Resilience – XR Glasses

Table 11: Reinforcement Learning Models Endpoints

5.2.5.3 TESTING, AND EVALUATION

The following results summarise component tests for the six rule-based reinforcement-learning (RB-RL) models, including interface validation, deterministic behaviour, boundary handling and agreement with the specified action rules.

<i>Category</i>	<i>Result (passed/failed)</i>
Infrastructure (registration, payload validation, invalid inputs/features rejection...)	33 / 0
Response schema (presence of all 8 output fields, type/range verification, no nulls)	18 / 0
Behavior (latency under threshold [$< 3.0s$], 10 parallel concurrent requests)	12 / 0
Boundary tests (main cell, neighbor gNB2/gNB3, UPFs switching thresholds)	60 / 0
Equivalence partitioning (combinatorial state, quality...)	4581 / 0

Table 12: Results of Reinforcement Learning Models Testing

The table reports 4,704 passes and no failures across the listed categories. This establishes the reported test outcomes for those cases. Agreement with a rule-derived test oracle does not establish an optimal recovery policy, and a latency check below 3 seconds at ten concurrent requests is not an end-to-end latency distribution or a production capacity limit.

5.2.5.4 MODEL DEPLOYMENT AND INFERENCE INFRASTRUCTURE

To manage model deployment and access, KServe was implemented on a dedicated server, sharing the same underlying infrastructure as the ALoTW model. Each of the six models is exposed as an independent HTTP/REST endpoint under the base service URL.

To manage model deployment and access, KServe was implemented on a dedicated server, sharing the same underlying infrastructure as the aLoTW model. Each of the six models is exposed as an independent HTTP/REST endpoint under the base service URL.

Each time a prediction is requested, the execution flow is as follows:

1. **Model selection:** routes the request to the loaded artefact for the requested flavour and device type.
2. **Pre-Process Phase:** Sanitizes and validates the 38-parameter input vector structure. The first 31 parameters are converted into the agent's categorical state (null, good, or high), while the remaining 7 QoS parameters are verified as mandatory—if any are missing or invalid, execution is immediately aborted with an HTTP 400 error.

3. **Predict Phase:** Processes the incoming metric state vector through the model to infer and generate the target set of 8 execution parameters.

5.2.5.5 POLICY INTERPRETATION AND ACTION ADMISSIBILITY

With gamma set to zero, the Q-value update moves the stored value towards the immediate reward observed for the selected state and action [20]. For the reported policy, this makes the reward definition central: the table expresses preferences encoded by the reward and categorical state representation. It does not establish that the selected action will minimise a sequence of future disruptions. A future study of delayed consequences would need an environment and reward design that represent those consequences explicitly.

The reward specification should describe how it balances the requested flavour, QoS continuity, resource use and the cost of reconfiguration. It should also identify prohibited actions, ties between equally valued actions and the treatment of unavailable target resources. These details explain why a particular configuration is selected and make the policy reproducible. They should accompany the model artefact rather than be inferred from endpoint names.

Before sending a proposal for approval, the vApp should validate the complete output vector against the feature and configuration schema. Checks should cover supported QoS values, the consistency of the SSC fields, existence of the requested slice and UPF, and the availability of any application-session information. At execution time, the nApp should recheck state-dependent preconditions because resources may have changed while CoCo was resolving the proposal.

Categorisation creates boundaries at which a small measurement change can switch the selected action. Validation should therefore inspect transitions, not only isolated states. Replaying observations immediately above and below each threshold can reveal repeated UPF switching, alternating QoS profiles or unnecessary resource recreation. The intended treatment of such oscillations should be part of the policy, with a measured trade-off between stability and reaction time.

5.2.6 OUTPUTS AND MESSAGE CONTRACT

On each periodic cycle, the vApp reports an intermediate outcome to CoCo, carrying the session and user identifiers, the action, the flavour, the achievable score, and the list of candidate actions for the session.

When CoCo grants permission, returning a final resolved score, the vApp correlates the response by session identifier, reuses the candidate actions from its own preceding intermediate outcome, and sends a UE configuration request to the nApp, using the same session identifier as the message identifier, together with the client identifier and the mapped UE configuration. When CoCo does not grant permission, it simply withholds the final resolved score; the vApp takes no further action for that cycle and does not call the nApp.

The vApp also acknowledges the local-ai-agent's start and stop requests, confirming that monitoring has been enabled or disabled for the session.

5.2.7 LIMITATIONS

The six model interfaces and their reported test campaign are described in Section 5.2.5. The release used by a running vApp, the active model versions and the production publication flag require deployment-specific evidence. The monitoring request uses a field named aLoTW for a value described semantically as cLoTW; this mapping needs to be explicit. Handling of multiple candidate actions also needs a single documented contract, including whether the model returns one configuration vector or a list of alternatives, and which proposal a CoCo approval authorises.

5.2.8 FAILURE HANDLING

Without a matching CoCo approval, the vApp does not request enforcement. The absence of a final resolved score is not an explicit rejection. A timeout, pending-proposal expiry and response correlation policy are therefore needed to distinguish unresolved requests from approvals and to prevent stale execution; these are validation and hardening requirements, not confirmed behaviour of the current implementation.

5.3 NETWORK APPLICATIONS (nAPP)

The nApp constitutes the enforcement layer of the Resilience Trust Function. Unlike the local-ai-agent and the vApp, it does not assess resilience trust levels or generate candidate actions. Instead, it receives the configuration approved through the CoCo-controlled workflow, translates it into the corresponding core-network operations, applies the required changes through the CumuCore API, and reports the execution outcome. This section describes its role, implementation stack, message interfaces, enforcement logic, outputs, limitations, and failure-handling behaviour.

5.3.1 ROLE AND POSITION IN THE RESILIENCE WORKFLOW

The nApp is the enforcement layer of the Resilience Trust Function. It receives, from the vApp, the UE configuration to apply once CoCo has granted permission for a session's resilience actions, applies the corresponding changes to the core network, and reports the outcome back to CoCo. Unlike the local-ai-agent and the vApp, the nApp does not itself assess resilience trust or propose actions; its responsibility is limited to enforcement and outcome reporting.

5.3.2 IMPLEMENTATION STACK

The nApp is implemented as a containerised service. It communicates with CoCo over an asynchronous message broker, exposes and consumes Agent-to-Agent (A2A) protocol tasks, and discovers and authorises access to the CumuCore API through OpenCAPIF. [9]

5.3.3 INPUTS AND MESSAGE CONTRACT

The nApp's triggering input is a request from the vApp to update the UE configuration for a session, carrying the session identifier, used as the message identifier, a client identifier, and the UE configuration to apply: the requested 5G QoS Identifier (5QI); Allocation and Retention Priority (ARP) level, capability, and vulnerability; Session and Service Continuity (SSC) default and allowed modes; the default slice identifier; whether an application session with QoS is requested; and the target user-

plane function. The nApp exposes this request as an Agent-to-Agent (A2A) task, `update_UE_configuration`, and as an equivalent REST endpoint; a corresponding read-only variant allows the current configuration to be retrieved without applying changes.

5.3.4 DECISION AND ENFORCEMENT LOGIC

On receiving a configuration request, the nApp evaluates, in order, four determinations against the session's current state in CumuCore: whether an existing application session with QoS should be deleted, because it is no longer requested and a valid existing subscription record exists; whether one should instead be created, because it is newly requested and no matching subscription exists, or the UE's IP address or application server identifier has changed; whether the UE's serving slice or user-plane function should be reassigned, based on either an explicit change in the requested default slice or an explicit request to switch between the two configured slices; and whether the UE's subscription profile, comprising its QoS identifier, priority and preemption attributes, and mobility mode settings, differs from what is requested and therefore needs to be updated. Whichever of these changes are applied, the nApp additionally creates a short-lived preemption configuration record in CumuCore, valid for twenty seconds from the time of the change, on a best-effort basis that does not block the request if it fails.

The table below summarises the four possible actions and their trigger conditions.

#	Action	Trigger condition
1	Delete application session with QoS	No longer requested, and a valid existing subscription record exists
2	Create application session with QoS	Newly requested and no matching subscription exists, or the UE's IP address or application server identifier has changed
3	Reassign slice or user-plane function	An explicit change in the requested default slice, or an explicit request to switch between the two configured slices
4	Update subscription profile	The UE's QoS identifier, priority/preemption attributes, or mobility mode settings differ from what is requested

Table 13: Action model endpoints

The twenty-second preemption record is an implementation-specific supporting object. Its creation is best-effort and does not establish that a preemption event occurred or that a recovery objective was met.

This rule-based logic operates entirely against the CumuCore API; discovery and authorisation is performed through OpenCAPIF. The reference architecture's 6G Core API, accessed through the Network Exposure Function (NEF), is realised in the current implementation through this same CumuCore API, rather than through a separate NEF-mediated system.

No machine learning model is used by the nApp: its enforcement logic is entirely deterministic and rule-based, comparing the requested configuration against the session's current state and applying only the differences.

QoS and mobility parameters need to be interpreted according to their defined semantics. 5QI identifies QoS characteristics; ARP concerns admission and retention, including preemption behaviour. Their numerical values should not be treated as a common resilience scale [3, clause 5.7]. The implementation-specific twenty-second preemption object is also distinct from demonstrating that standardised ARP behaviour has been exercised.

An update accepted by the subscription API can affect subsequent session establishment without immediately changing an active PDU session. In particular, changing default or allowed SSC modes is not evidence that the established session has switched SSC mode [3, clause 5.6.9]. Integration tests should record the applicable activation procedure and any UE reconnection, session re-establishment or traffic interruption needed for each supported change.

5.3.5 OUTPUTS AND MESSAGE CONTRACT

The nApp returns the outcome of processing the requested changes described in Section 5.3.4. When the originating request includes both session and user identifiers, it also sends a best-effort success or failure report to CoCo. The report should be interpreted as an execution outcome; effective state and subsequent service behaviour require separate verification.

5.3.6 LIMITATIONS

The rule catalogue described in Section 5.3.4 covers only core-configuration-style actions (subscription, session, and slice/user-plane function changes); no separate rule category for fault tolerance and recovery, networking, or monitoring and logging actions was identified in the code reviewed, notwithstanding these being informally described as part of the function's purpose (Section 3). The separate 6G Core API execution path via NEF shown in the reference architecture is not implemented as a distinct system in the current codebase; the nApp realises this function entirely through the CumuCore API.

5.3.7 FAILURE HANDLING

The nApp's outcome reporting to CoCo is explicitly best-effort and does not affect its response to the vApp: a failure to notify CoCo of an outcome would not, by itself, be visible to the component that requested the change. The short-lived preemption configuration record created alongside a configuration change is created on the same best-effort basis and does not block the request if it fails (Section 5.3.4).

6 TRUST EVALUATION & ENFORCEMENT LOGIC

6.1 DATA ACQUISITION AND INDICATOR CALCULATION

The function combines CoCo trust messages, CumuCore context and configuration state, Prometheus radio metrics, and local per-gNB capacity and frequency configuration. CumuCore access is discovered and authorised through OpenCAPIF. Data acquisition occurs at initial assessment and on subsequent vApp monitoring cycles. MLOps consumes the resulting feature vector to provide the score or action prediction.

From the raw Prometheus counters, success-rate percentages are derived for radio resource control (RRC) connection establishment, handover, and initial UE context setup. Registered- and connected-UE occupancy ratios are derived from CumuCore's reported current UE counts relative to the gNB's maximum registered and connected UE counts, drawn from a local configuration table. The gNB's serving frequency band, also drawn from this table, is mapped to a coarse frequency category, which the vApp additionally uses, independently of any model, as a deterministic input to a static frequency-to-flavour lookup. These indicators, together with the non-calibrated resilience level (nLoTW) requested by CoCo, form the basis for the resilience assessment.

6.2 TRUST EVALUATION

The local-ai-agent uses the device-specific score predictor to estimate aLoTW at session start. During periodic monitoring, the vApp uses a flavour-specific action predictor. Because the action predictor's stated output is a configuration vector, it does not by itself demonstrate a newly calculated aLoTW. The source and freshness of the score included in periodic outcome messages must therefore be defined separately.

The local-ai-agent reports its computed aLoTW to CoCo as the Resilience Trust Function's contribution to the system-wide trust computation, together with the resilience flavour. CoCo reconciles this against the level originally requested and against the other Trust Functions' contributions, returning a calibrated resilience trust level (cLoTW) together with a start or stop instruction for the session. On each subsequent periodic cycle, the vApp reports an intermediate outcome, comprising an achievable score, a flavour, and a list of candidate actions, and CoCo returns a final, calibrated resolved score only when it grants permission for those actions. Consistent with the scope agreed for this deliverable, how CoCo itself derives the calibrated value from the Resilience Trust Function's contribution and those of the other Trust Functions is not described here. No confidence value distinct from the aLoTW score itself was identified in the material reviewed.

6.3 ENFORCEMENT LOGIC

When a final resolved score authorises a pending proposal, the vApp correlates it with the session and maps the stored candidate actions into a UE configuration request. The nApp compares that request with the current CumuCore state and determines which application-session, slice or UPF, and

subscription-profile changes are needed. A proposal that has expired or no longer matches the current context should be reassessed before execution; Section 4.8 describes this design requirement.

Enforcement takes place entirely at the nApp and is hard enforcement: the identified configuration changes, subscription profile updates, application-session-with-QoS creation or deletion, and slice or user-plane function reassignment, are applied directly to the core network through the CumuCore API, rather than issued as a recommendation left to another party to act on. No soft-enforcement mechanism, such as an advisory recommendation, warning, or prioritisation suggestion that does not itself change the network configuration, was identified for the Resilience Trust Function in the material reviewed; this asymmetry is noted as a design characteristic to revisit in Section 7. Whenever a configuration change is applied, a short-lived preemption configuration record, valid for twenty seconds, is additionally created in CumuCore on a best-effort basis.

6.4 FEEDBACK, REASSESSMENT, AND FALLBACK HANDLING

After applying or attempting a configuration change, the nApp reports the outcome, success or failure, back to CoCo on a best-effort basis, provided the originating request carried both a session and a user identifier; this report does not alter the nApp's direct response to the vApp. When CoCo does not grant permission for a proposed action, it simply withholds the final resolved score for that cycle; neither the vApp nor the nApp takes further action, and no explicit rejection or reassessment trigger is defined. The vApp's periodic cycle then repeats the assessment for the same session at the next interval, for as long as the session remains under active monitoring, until CoCo issues a stop instruction, through the same resolved-score message type used to start monitoring, identifying the session by the same UE identity used at start.

Each layer defines its own fallback for degraded conditions. At the local-ai-agent, if the model-serving call cannot be completed, a deterministic rule adjusts the requested resilience level by a fixed, device-type-dependent amount, bounded to a valid range, rather than leaving the session unassessed. At the vApp, if the action-mapping step cannot be completed, a deterministic fallback constructs a configuration request that preserves the session's current state in CumuCore, avoiding an unintended change. At the nApp, the supporting preemption-configuration record and the outcome report to CoCo are both created on a best-effort basis and do not block or alter the primary configuration response if they fail. The behaviour of the local-ai-agent and the vApp when telemetry retrieval itself, from Prometheus or CumuCore, is incomplete or fails has not been confirmed (Sections 5.1.7 and 5.2.7) and is not asserted here.

6.5 TELEMETRY QUALITY AND TIMING

A decision should use an identifiable measurement snapshot. Each input should carry, directly or through its query context, the observed entity, measurement interval and acquisition time. A mixture of fresh UE configuration and old cell load can describe a network state that never existed. The acceptable age and time skew should therefore be set for each feature family and validated against the response time expected from the control loop.

For cumulative event counters, success ratios should compare successes and attempts over the same time window. Prometheus rate and increase functions account for counter resets; rate should be applied before aggregation so that resets remain detectable [15]. Utilisation gauges and already calculated percentages require different processing. An empty query result or a stale series should not be silently converted into zero, because that would confuse missing evidence with an observed measurement [16].

A zero-attempt interval provides no basis for a success percentage. The feature contract should specify whether that condition produces a missing value, an explicit availability flag, or a retained observation subject to an age limit. The same rule must apply during training and inference. This matters particularly for optional neighbours: absence of a neighbouring gNB is a different condition from a reachable neighbour with zero load.

As an illustrative calculation, 98 successful attempts out of 100 yield a 98% success ratio. Under the stated RRC threshold of greater than 95%, that observation belongs to the good category; exactly 95% belongs to the high-risk category. Similarly, 40 connected UEs against a configured capacity of 50 yield 80% occupancy, which belongs to the high category. These examples explain boundary behaviour and are not measurements from a SAFE-6G deployment.

6.6 VERIFICATION OF EXECUTION AND SERVICE EFFECT

Three outcomes should be recorded separately: acceptance of the request by the core API, observation of the effective configuration, and observation of the intended service effect. The existing nApp response and CoCo outcome message primarily address the first. A stronger verification loop would read back the affected objects and then evaluate the UE over a defined observation interval, accounting for any delay before the change becomes active.

Before a change, the controller should retain the relevant configuration and a small set of session indicators. Afterward, it should compare the effective state with the approved target and assess continuity, packet delivery or application-level behaviour using the indicators chosen for the pilot. The observation interval must be long enough to capture the effect while still allowing a failed action to be detected promptly. The interval and thresholds are validation parameters, not universal constants.

A multi-step update can succeed partially. For example, deletion of an application session could succeed before creation of its replacement fails. The response should preserve the outcome of each attempted step and the state subsequently observed. Retrying the entire request without inspecting that state can repeat side effects. A compensation or escalation policy should identify which changes can be reversed and which require a new decision from CoCo; automatic rollback is not established by the current description.

Attribution also matters. A service can recover because background load decreases or another controller repairs a fault. A controlled comparison should therefore use the same disturbance and similar initial conditions for the evaluated policy and its baseline, and retain a timeline of other changes. Configuration success and improvement observed after configuration are both useful evidence, but neither alone proves that the policy caused the improvement.

6.7 ILLUSTRATIVE SESSION WORKFLOW

Consider an XR session whose serving cell becomes congested while an alternative UPF remains available. The vApp collects the current context and invokes the model for the session's approved flavour. If the model proposes a UPF or QoS change, the vApp validates its fields and sends the candidate to CoCo. Until a matching approval arrives, the session retains its existing configuration. This example illustrates the specified division of responsibilities without prescribing a particular model output.

After approval, the nApp rereads the relevant state and applies only the required changes. If the target already matches the current state, the outcome should record that no configuration change was needed. If the target resource has disappeared, the action should fail its precondition and return for reassessment. The proposed verification stage then checks effective configuration and the chosen service indicators before treating the resilience objective as satisfied.

7 FUTURE RESEARCH PATHWAYS

The User-centric Resilience Trust Function developed within SAFE-6G establishes a framework for per-session resilience assessment, AI-assisted decision-making, CoCo-mediated coordination, and core-network enforcement. Building on this foundation, several research and engineering pathways can further extend the capabilities of the function beyond the scope addressed in the current implementation.

7.1 EVOLUTION OF THE CLOSED-LOOP RESILIENCE WORKFLOW

A natural evolution of the current control loop is to enrich the interaction between assessment, approval, enforcement, and post-action observation. The existing implementation performs direct core-network reconfiguration following CoCo approval and reports the corresponding execution outcome. Future work could complement this mechanism with additional execution modes, including advisory actions, explicit approval or rejection responses, and richer feedback about the effective configuration observed after enforcement.

The enforcement catalogue could also be extended beyond the currently supported subscription, application-session, slice, and UPF configuration actions. Additional mechanisms could incorporate infrastructure-level recovery, RAN-related resilience actions, and coordinated failover strategies where these capabilities are exposed by the underlying network platform.

Further research direction is the refinement of the temporal aspects of the control loop. This includes characterizing the freshness of periodic aLoTW assessments, defining appropriate validity periods for proposed actions, and evaluating how quickly approved configuration changes become effective for active UE sessions.

7.2 ADVANCED INTELLIGENCE AND USER-AWARE RESILIENCE

The current AI-assisted approach provides a basis for exploring more advanced resilience intelligence. Future research could investigate the integration of explainability mechanisms from the wider SAFE-6G MLOps framework, allowing the achievable resilience score and the actions proposed by the Resilience Trust Function to be accompanied by information about the network conditions and model factors that influenced the decision.

Another promising direction is to extend the user-centric dimension beyond device type. The current approach differentiates between device categories and network conditions; future models could additionally consider application characteristics, service requirements, user preferences, or context-specific resilience objectives.

Cross-Trust-Function coordination through CoCo also provides opportunities for further research. In particular, future work could study how resilience decisions interact with Safety, Security, Privacy, and Reliability requirements under situations in which different trust dimensions recommend competing network configurations.

The robustness of the distributed decision workflow could also be further strengthened through explicit handling of duplicated, delayed, reordered, or temporarily unavailable messages across Kafka and Agent-to-Agent interactions.

7.3 ARCHITECTURAL EVOLUTION AND INTEROPERABILITY

The current implementation relies on OpenAPIF for API discovery and authorisation, CumuCore for core-network context and configuration, MLOps for model lifecycle and serving, and Prometheus for network telemetry. These integrations provide a practical implementation environment while also offering opportunities for architectural evolution.

Future work could evaluate additional core-network platforms and more standardised exposure mechanisms, including deployments in which the 6G Core API is accessed through an explicit NEF-mediated path. This would make it possible to assess the portability of the Resilience Trust Function across different core network implementations while preserving the component interfaces defined in SAFE-6G.

Additional degraded-operation strategies could also be investigated for temporary unavailability of telemetry, API-discovery, or network-context services, complementing the fallback mechanisms already defined for model-serving and action-mapping operations.

7.4 VALIDATION, SCALABILITY, AND OPERATIONALISATION

The validation methodology defined in SAFE-6G provides a basis for progressively evaluating the Resilience Trust Function under increasingly representative conditions. Future validation can extend the current synthetic model evaluation towards operational network observations, including the study of distribution drift and the relationship between predicted achievable resilience and the service behaviour observed after an action.

This evaluation can distinguish complementary dimensions such as prediction quality, action-execution success, calculation and recommendation latency, uptime, time to provide a resilience level, and the resulting effect on the UE service. The Resilience Function Success Rate and the additional KPIs defined in D2.4 [6] provide a basis for this broader assessment.

Scalability represents another relevant evolution pathway. The current deployment provides a simple baseline for per-session monitoring; future deployments could evaluate multiple replicas, persistent monitoring state, session ownership and reassignment, and mechanisms to avoid duplicated evaluations or competing configuration actions when the function is horizontally scaled.

7.5 APPLICATION-AWARE RESILIENCE AND INTERACTION WITH THE METAVERSE ENVIRONMENT

A further evolution pathway for the Resilience Trust Function is to extend its user-centric control loop towards the application plane by leveraging the Metaverse Manager interfaces already available within the SAFE-6G architecture. The Metaverse Manager acts as the integration point between the

SAFE-6G framework and the XR use-case applications, exposing APIs through OpenCAPIF that allow authorised SAFE-6G components to retrieve application information, monitor the state of the running use cases, and trigger actions within the virtual environment.

In particular, the Metaverse Manager provides access to real-time performance metrics from the XR headsets through its monitoring interfaces, enabling future resilience assessments to correlate radio and core-network conditions with application-level indicators and the service actually experienced by the user.

The interaction can also operate in the opposite direction. Following a resilience decision or an enforced network reconfiguration, the Resilience Trust Function could invoke application-facing capabilities exposed by the Metaverse Manager. This could be used, for example, to notify the XR environment of the resulting resilience action or to coordinate application-level behaviour with network-level adaptation. Existing Metaverse APIs already demonstrate this interaction pattern by enabling SAFE-6G components to trigger actions in the virtual environment and to provide information to users through the XR interface.

As an initial application-facing integration scenario, this mechanism could also be used to notify the XR environment when a resilience-related network configuration change has been successfully processed, allowing the resulting status to be presented to the end user.

Such an extension would provide a path towards a broader end-to-end resilience loop in which network state, core-network reconfiguration and application-level observations are considered together. It would also align with the UC1 requirement for maintaining suitable and continuous XR service under degraded conditions. The definition of resilience-specific application of actions and their integration into the decision and enforcement workflow constitute a natural evolution beyond the current scope of the Resilience Trust Function.

8 CONCLUSIONS

The User-centric Resilience Trust Function is one of five Trust Functions within the SAFE-6G trustworthiness framework, addressing the resilience dimension of trustworthiness: the capacity of a user equipment (UE) session and its associated core network configuration to withstand, adapt to, and recover from degraded conditions or connection faults. It is architecturally distributed across three cooperating layers, the local-ai-agent, the vApp, and the nApp, responsible respectively for assessing achievable resilience, orchestrating session-level decisions and continuous monitoring, and enforcing the resulting configuration on the core network.

Its principal inputs are the non-calibrated resilience trust level requested by the Cognitive Coordinator (CoCo) and the telemetry gathered from the core network and radio access infrastructure; its principal outputs are the achievable resilience trust level (aLoTW) it reports to CoCo and the core network configuration changes it applies once CoCo grants permission. The function contributes to the overall Level of Trustworthiness (LoTW) framework by reporting its achievable resilience assessment to CoCo, which reconciles it against the level originally requested and against the assessments of the other Trust Functions, returning a calibrated resilience trust level (cLoTW) that the function then uses both to determine enforcement actions and to start or stop active monitoring of a session.

Within the broader SAFE-6G trustworthiness framework, the Resilience Trust Function operates as one of the coordinated, AI-assisted mechanisms through which CoCo translates user-centric trust requirements into network behaviour, alongside the Safety, Security, Privacy, and Reliability Trust Functions. Its relevance to user-centric trust lies in evaluating and enforcing resilience at the level of an individual UE session, tailored to the user's device type and current network conditions, rather than applying a uniform, network-wide resilience policy.

Building on the architecture and implementation presented in this deliverable, the Resilience Trust Function provides a foundation for continued evolution towards increasingly comprehensive user-centric resilience. The future research pathways identified in Section 7 include richer post-action observation and feedback, broader resilience mechanisms, enhanced AI-assisted decision-making, scalability across distributed deployments, and interaction with the application plane through the SAFE-6G Metaverse environment. These directions extend the capabilities developed within SAFE-6G beyond the current implementation and provide a basis for future research towards coordinated end-to-end resilience across the application, network, and infrastructure layers.

9 REFERENCES

- [1] SAFE-6G Consortium, D4.1 Cognitive coordinator, AI agents and user-centric functions, v1.1, Section 8. https://safe-6g.eu/wp-content/uploads/2026/02/D4.1_SAFE-6G_v1.1.pdf
- [2] SAFE-6G Consortium, D2.1 Definition of Technical Requirements for user-centric 6G Trustworthiness, v1.1, Section 5.7.2. https://safe-6g.eu/wp-content/uploads/2026/02/D2.1_SAFE-6G_v1.1.pdf
- [3] 3GPP, TS 23.501 System architecture for the 5G System (5GS), v18.8.0, Release 18, ETSI TS 123 501, January 2025. See clauses 5.6.9 and 5.7. https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/18.08.00_60/ts_123501v180800p.pdf
- [4] 3GPP, TS 32.541 Telecommunication management; Self-Organizing Networks (SON); Self-healing concepts and requirements, v18.0.0, Release 18, ETSI TS 132 541, May 2024. https://www.etsi.org/deliver/etsi_ts/132500_132599/132541/18.00.00_60/ts_132541v180000p.pdf
- [5] ETSI, GS NFV-REL 001 Network Functions Virtualisation (NFV); Resiliency Requirements, v1.1.1, January 2015. https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_nfv-rel001v010101p.pdf
- [6] SAFE-6G Consortium, D2.4 6G Trustworthiness KPI & KVI definition and validation methodology, v1.1, Section 4.1.2.4, Table 23 and Section 5.3.3. https://safe-6g.eu/wp-content/uploads/2026/02/D2.4_SAFE-6G_v1.1.pdf
- [7] SAFE-6G Consortium, local-ai-agent source code repository, GitLab. Implementation reference. <https://gitlab.com/safe-6g/development/resilience-function/local-ai-agent>
- [8] SAFE-6G Consortium, vApp source code repository, GitLab. Implementation reference. <https://gitlab.com/safe-6g/development/resilience-function/vapp>
- [9] SAFE-6G Consortium, nApp source code repository, GitLab. Implementation reference. <https://gitlab.com/safe-6g/development/resilience-function/napp>
- [10] 3GPP, TS 23.288 Architecture enhancements for 5G System (5GS) to support network data analytics services, v18.8.0, Release 18, ETSI TS 123 288, January 2025. https://www.etsi.org/deliver/etsi_ts/123200_123299/123288/18.08.00_60/ts_123288v180800p.pdf
- [11] ETSI, GS ZSM 009-1 Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 1: Enablers, v1.1.1, June 2021. https://docbox.etsi.org/ISG/ZSM/Open/Publications_pdf/ZSM%20009-1v1.1.1%20-%20GS%20-%20Closed-loop%20autom%20enablers.pdf
- [12] 3GPP, TS 23.222 Common API Framework for 3GPP Northbound APIs, v18.4.0, Release 18, ETSI TS 123 222, April 2024. https://www.etsi.org/deliver/etsi_ts/123200_123299/123222/18.04.00_60/ts_123222v180400p.pdf

- [13] 3GPP, TS 33.122 Security aspects of Common API Framework (CAPIF) for 3GPP northbound APIs, v18.4.0, Release 18, ETSI TS 133 122, July 2024. https://www.etsi.org/deliver/etsi_ts/133100_133199/133122/18.04.00_60/ts_133122v180400p.pdf
- [14] ETSI SDG OpenCAPIF, Interfaces. Online documentation, accessed 23 September 2026. <https://ocf.etsi.org/interfaces/>
- [15] Prometheus Authors, Query functions, Prometheus 2.55 documentation, rate and increase. Accessed 23 September 2026. <https://prometheus.io/docs/prometheus/2.55/querying/functions/>
- [16] Prometheus Authors, Querying basics, including staleness and time-series selection. Accessed 23 September 2026. <https://prometheus.io/docs/prometheus/latest/querying/basics/>
- [17] L. Breiman, Random Forests, Machine Learning, vol. 45, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324. <https://doi.org/10.1023/A:1010933404324>
- [18] scikit-learn developers, RandomForestRegressor API reference. Accessed 23 September 2026. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>
- [19] scikit-learn developers, Metrics and scoring: quantifying the quality of predictions, regression metrics. Accessed 23 September 2026. https://scikit-learn.org/stable/modules/model_evaluation.html
- [20] C. J. C. H. Watkins and P. Dayan, Q-learning, Machine Learning, vol. 8, pp. 279–292, 1992. DOI: 10.1007/BF00992698. <https://doi.org/10.1007/BF00992698>
- [21] P. Auer, N. Cesa-Bianchi and P. Fischer, Finite-time Analysis of the Multiarmed Bandit Problem, Machine Learning, vol. 47, pp. 235–256, 2002. DOI: 10.1023/A:1013689704352. <https://doi.org/10.1023/A:1013689704352>
- [22] KServe contributors, Inference Protocol V1. Accessed 23 September 2026. <https://kserve.github.io/website/docs/concepts/architecture/data-plane/v1-protocol>
- [23] Apache Software Foundation, Apache Kafka 4.0 Design, Message Delivery Semantics. Accessed 23 September 2026. <https://kafka.apache.org/40/design/design/>
- [24] Kubernetes Authors, Liveness, Readiness, and Startup Probes. Accessed 23 September 2026. <https://kubernetes.io/docs/concepts/workloads/pods/probes/>
- [25] Kubernetes Authors, Horizontal Pod Autoscaling. Accessed 23 September 2026. <https://kubernetes.io/docs/concepts/workloads/autoscaling/horizontal-pod-autoscale/>