

SAFE-6G

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

Deliverable D4.2: Cognitive Coordinator and (X)AI Aggregator of Human-Centric 6G Trustworthiness

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	D4.2
Document title	Cognitive Coordinator and (X)AI Aggregator of Human-Centric 6G Trustworthiness
Lead Beneficiary	THALES
Editor(s)	Stephane Lorin (THALES) Dimitrios Roidis (NCSRD)
Author(s)	Dimitrios Roidis (NCSRD) Spyridon Georgoulas (NCSRD) Harilaos Koumaras (NCSRD) Stephane Lorin (THALES) Kevin Philippe (THALES) Vaios Koumaras (INF) Evangelos Anagnostopoulos (INF)
Dissemination level	Public
Contractual date of delivery	30/09/2026
Status	Final
File name	SAFE-6G_D4.2_V1.0.pdf

Revision History

Version	
V0.1	Initial ToC
V0.2	Major CoCo Contributions
V0.3	Major xAI Contributions
V0.4	Figure and Table Captions, Homogenization of Content
V0.5	Version Sent for Internal Review
V0.6	Address of Internal Review Comments
V1.0	Editing and Finalization

GLOSSARY

Abbreviations/Acronym	Description
ALE	Accumulative Local Effects
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CAIO	Cognitive AI Orchestrator
CEM	Constructive Explanation Method
cLoTw	Calibrated Level of Trustworthiness
CoCo	Cognitive Coordinator
CRM	Conflict Resolution Manager
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DT	Digital Twin
GET	HTTP Get Method
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
JSON	Javascript Object Notation
LLM	Large Language Model
LoTw	Level of Trustworthiness
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MSE	Mean Square Error
MSISDN	Mobile Station International Subscriber Directory number
nLoTw	Normalized Level of Trustworthiness
NLP	Natural Language Processing
NLTK	Natural Language ToolKit
POST	HTTP Post Method
REST	Representational State Transfer
RMSE	Root Mean Square Error
SHAP	SHapley Additive exPlanations
SUPI	Subscription Permanent Identifier
SVM	Support Vector Machine
TF	Trust Function
TF-IDF	Term Frequency – Inverse Document Frequency
URI	Uniform Resource Identifier
xAI	Explainable AI
XR	Extended Reality

EXECUTIVE SUMMARY

This deliverable presents the architectural design and development of the Cognitive Coordinator (CoCo) and eXplainable AI (XAI) module, building upon the SAFE-6G reference architecture established in D2.2 and the initial architectural and implementation work presented in D4.1. Together, these components provide the cognitive coordination and explainability capabilities of the SAFE-6G trustworthiness framework, supporting the interpretation of user-defined trustworthiness requirements, the coordination of the five Trust Functions (Safety, Security, Privacy, Resilience and Reliability) and the explanation of the decisions produced throughout the trustworthiness provision lifecycle.

The Cognitive Coordinator is responsible for managing the end-to-end processing of user requests, translating the trustworthiness requirements expressed through the SAFE-6G Chatbot into quantitative trustworthiness levels and coordinating their evaluation and enforcement through the involved Trust Functions. It considers the session and user context, reconciles potentially incompatible trustworthiness configurations, and determines the final calibrated Level of Trustworthiness (cLoTw). CoCo subsequently coordinates the corresponding deployment actions and collects their outcomes, maintaining a consistent processing flow from the initial user request to the final response.

The XAI module, while developed as a distinct component with its own functionality and interfaces, remains closely associated with CoCo and its trustworthiness provision lifecycle. It supports the interpretability of the AI-driven decisions produced by CoCo and the Trust Functions involved in each user request. By providing explanations for the resulting trustworthiness decisions, XAI contributes to the framework's transparency and enables users to better understand how their requirements have been interpreted and addressed.

Beyond presenting the functional architecture of Cognitive Coordinator and the per component role in the trustworthiness provision lifecycle, this deliverable documents the principal development activities, implementation decisions and component interactions adopted for CoCo and XAI. It demonstrates how the architectural concepts established in earlier SAFE-6G deliverables have been translated and evolved into developed components supporting an end-to-end trustworthiness provision flow. These developments provide the technical basis for the subsequent validation of trustworthiness coordination and explainability within the SAFE-6G experimentation environments and use cases.

KEYWORDS

Cognitive Coordinator, xAI aggregator, trustworthiness, AI, ML models

TABLE OF CONTENTS

1	<i>Introduction.....</i>	<i>1</i>
1.1	Rationale Behind This Deliverable.....	1
1.2	Technical Requirements	1
2	<i>Cognitive Coordinator – Architecture And Functional Design</i>	<i>3</i>
2.1	Interface With chatbot	4
2.2	Cognitive AI Orchestrator	5
2.2.1	CoCo Graphical User Interface	6
2.2.2	User Registry	6
2.2.3	Digital Twin Prediction Module	6
2.3	Machine Learning Module	7
2.4	Message Broker	8
2.5	Reasoning Engine	9
2.5.1	Knowledge Base	9
2.5.2	Conflict Resolution Manager And Service Function Scheduling	10
2.6	Explainable AI	10
3	<i>Cognitive Coordinator Lifecycle</i>	<i>12</i>
4	<i>Cognitive Coordinator – Implementation and Development Desicions</i>	<i>14</i>
4.1	Interface With Chatbot.....	14
4.2	Cognitive AI Orchestrator	15
4.2.1	CoCo Graphical User Interface	17
4.2.2	User Registry	20
4.2.3	Digital Twin Prediction Module	21
4.3	Machine Learning Module	24
4.3.1	Training	24
4.3.2	Inference.....	28
4.4	Message Broker	30
4.5	Reasoning Engine	32
4.5.1	Knowledge Base	32
4.5.2	Conflict Resolution Manager and Service Function Scheduling	33
4.6	Explainable AI	34
4.6.1	Operational context	34
4.6.2	Structure of the XAI aggregator	35
4.6.3	Evaluation module.....	38
4.6.4	XAI Aggregator Deployment	38
4.7	Coco and xai as a whole – interactions and data flow	43

4.8	XAI requirements coverage.....	43
4.8.1	Collecting inputs for the XAI component	44
4.8.2	Building the XAI models.....	44
4.8.3	Sending results to the chatbot	45
4.8.4	Proposing different visualizations of the XAI results	45
4.8.5	Allowing real-time interactions with the end-user	46
5	<i>Future Evolution of CoCo and Alignment with 3GPP Intent-Driven Management</i>	47
5.1.1	From the SAFE-6G Implementation to a Generic Cognitive Coordinator	47
5.1.2	CoCo as a 3GPP-Aligned Intent Handling Function	47
5.1.3	Profile-Driven Architecture and Mapping to the SAFE-6G Implementation	49
5.1.4	Standards-Oriented Future Activities	50
6	<i>Conclusion</i>	52
7	<i>References.....</i>	53

List of FIGURES

Figure 1: Functional View of SAFE-6G Architecture as presented in D2.2.....	3
Figure 2: Evolved View of SAFE-6G User-Centric Cognitive Framework.....	4
Figure 3: SAFE-6G Chatbot	5
Figure 4: GUI, User Registry and Digital Twin's Involvement Constitute the Cognitive AI Orchestrator.....	5
Figure 5: Machine Learning Module of CoCo	7
Figure 6: Message Broker Component and its Interaction with the Trust Functions.....	8
Figure 7: Reasoning Engine, Constituted by the Knowledge Base, the Conflict Resolution Manager and Service and Function Scheduling	9
Figure 8: XAI Aggregator as part of CoCo.....	10
Figure 9: CoCo's Lifecycle View	12
Figure 10: Chatbot Payload Towards CoCo.....	14
Figure 11: CoCo's Response to the Chatbot after the Session is Finalized	15
Figure 12: Information Stored in a Snapshot, consumed by the GUI	18
Figure 13: CoCo's Graphical User interface Overview	19
Figure 14: GUI's Central Lifecycle View.....	19
Figure 15: Users entries stored in the User Registry	20
Figure 16: Spatial distribution of predicted RSRP values across the deployment area. The yellow dot indicates the antenna location	23
Figure 17: Spatial distribution of predicted SINR values across the deployment area. The yellow dot indicates the antenna location	24
Figure 18: ML model's training process	25
Figure 19: Validation and Training Loss per epoch	27
Figure 20: MAE and RMSE per epoch	27
Figure 21: R squared per epoch	28
Figure 22: ML model's Inference process	29
Figure 23: Bidirectional engagement of the Message Broker's communication with the Privacy TF ..	31

Figure 24: Knowledge Base populated with Conflict Rules, implemented in python	32
Figure 25: Reasoning Engine's workflow	33
Figure 26: XAI - Overview of Operational Context.....	35
Figure 27: XAI - Decomposition of Sub-Modules	35
Figure 28: XAI Selector Sub-Module	36
Figure 29: XAI Interpreter Sub-Module	38
Figure 30: XAI Aggregator sequence diagram.....	39
Figure 31: XAI - Orchestration Logic Sequence Diagram	43
Figure 32: Future positioning of CoCo as a 3GPP-aligned Intent Handling Function. The trustworthiness functionality developed in SAFE-6G represents one supported management profile alongside other service, RAN/edge, and 5G Core domains (adapted from [13])	48
Figure 33: Profile-driven CoCo architecture. Common intent lifecycle and reporting functions are decoupled from profile-specific cognitive models, reasoning policies, communication mechanisms, and Action Adapters (adapted from [5]).....	49

List of TABLES

Table 1: Cognitive Coordinator's Gitlab Repository.....	2
Table 2: Exchanged variables via the CoCo API	15
Table 3: Session information maintained by the CAIO	17
Table 4: CAIO APIs exposed to the GUI.....	18
Table 5: User Registry Database Schema and Column Descriptions	21
Table 6: Snippet of ML model's training dataset	25
Table 7: Metrics produced during the evaluation of ML model and its Baseline.....	26
Table 8: Topic names and Payloads produced and consumed by the Message Broker	31
Table 9: POST /explain request payload	40
Table 10: Model descriptor schema registered per domain	42
Table 11: Feature description schema.....	42
Table 12: POST /explain response	43
Table 13: Mapping between the current SAFE-6G CoCo implementation, the generic CoCo abstraction, and 3GPP intent-driven management concepts.....	50

1 INTRODUCTION

1.1 RATIONALE BEHIND THIS DELIVERABLE

This deliverable marks the transition of the SAFE-6G trustworthiness framework from the architectural definitions and initial implementations established in D4.1 towards the development and deployment of its core cognitive and trust-enabling components. In particular, it consolidates the implementation and deployment of the Cognitive Coordinator (CoCo) and the XAI Aggregator, building directly on the architectural foundations introduced in D4.1.

The purpose of this deliverable is to demonstrate how these architectural concepts are realized as operational and deployable components within the SAFE-6G framework, while documenting their internal design and functionality. Particular emphasis is placed on the AI-driven mechanisms that interpret user trustworthiness requirements, determine appropriate trustworthiness levels, support their enforcement through the Trust Functions, and provide explainability for the resulting decisions.

Accordingly, the deliverable first presents the overall architecture and internal interoperability of CoCo and the XAI Aggregator through a sequential, lifecycle-driven description, and subsequently details the implementation choices and technical developments adopted for the individual components. This provides the technical foundation for their validation within the SAFE-6G use cases and experimentation environments. Section 5.1 additionally discusses how the implemented CoCo mechanisms can be generalized beyond the trustworthiness domain and progressively aligned with 3GPP intent-driven management standards.

1.2 TECHNICAL REQUIREMENTS

The development and implementation activities related to the Cognitive Coordinator and the XAI Aggregator presented in this deliverable have been guided by the following requirements defined in D2.1:

Common Requirements

- **REQ-COM-RES-F-M-01:** Holistic trustworthy approach
- **REQ-COM-ETH-NF-M-05:** Conformance with the AI act
- **REQ-COM-RES-F-M-07:** Data availability
- **REQ-COM-DES-F-R-09:** Interface to MLOPS framework to train and execute ML models
- **REQ-COM-DES-F-M-11:** Supporting Infrastructure
- **REQ-COM-DES-F-M-12:** Interoperability Requirement
- **REQ-COM-RES-F-M-13:** Efficient AI models
- **REQ-COM-DES-F-R-24:** Fault Tolerance

Cognitive Coordinator Requirements

- **REQ-COG-REAS-F-R-01:** Trust Functions Profiling and Storage

- **REQ-COG-AISER-F-M-02:** AI-Assisted Learning and Adaptation
- **REQ-COG-AISER-F-M-03:** AI Prediction Accuracy and Metrics
- **REQ-COG-KBASE-F-M-04:** Advanced Knowledge Base That Supports The Reasoning Engine
- **REQ-COG-REAS-F-R-05:** Context-aware Reasoning that supports Trust Score Calculation
- **REQ-COG-AISER-F-M-06:** Accurate Feature extraction
- **REQ-COG-AISER-F-R-07:** Non-calibrated Output Validation
- **REQ-COG-AISER-F-M-08:** Explainable AI Models to Improve User's/Tenant's Level of Trust
- **REQ-COG-KBASE-NF-R-09:** Ensure Knowledge Base Integrity
- **REQ-COG-SCHED-F-M-10:** Near-Real time data processing and response
- **REQ-COG-SCHED-F-M-11:** Management of Cloud Computing continuum resources
- **REQ-COG-SCHED-F-M-12:** Service and Function Scheduling
- **REQ- COG -FRA-F-M-13:** Operational Processes Requirement
- **REQ-COG -PRO-F-M-14:** Programmable Cognitive Coordination

XAI Aggregator requirements

- **REQ-MLOPS-XAI-F-M-01:** Collect inputs for XAI component
- **REQ-MLOPS-XAI-F-M-04:** Build XAI models
- **REQ-MLOPS-XAI-F-M-05:** Send results to the chatbot
- **REQ-MLOPS-XAI-F-M-06:** Propose different visualizations of the XAI results
- **REQ-MLOPS-XAI-NF-M-08:** The XAI component should allow real-time interactions with the end-user

Source Code	https://gitlab.com/safe-6g/development/cognitive-coordinator
License	Apache 2.0

Table 1: Cognitive Coordinator's Gitlab Repository

2 COGNITIVE COORDINATOR – ARCHITECTURE AND FUNCTIONAL DESIGN

The Cognitive Coordinator (CoCo) is the central intelligence and coordination component of the SAFE-6G trustworthiness framework. As defined in the SAFE-6G reference architecture, CoCo interprets the trustworthiness requirements expressed by the user, derives the corresponding Level of Trustworthiness (LoTw) using its AI capabilities, and coordinates the Trust Functions responsible for realizing these requirements across the five trustworthiness dimensions of Safety, Security, Privacy, Resilience, and Reliability.

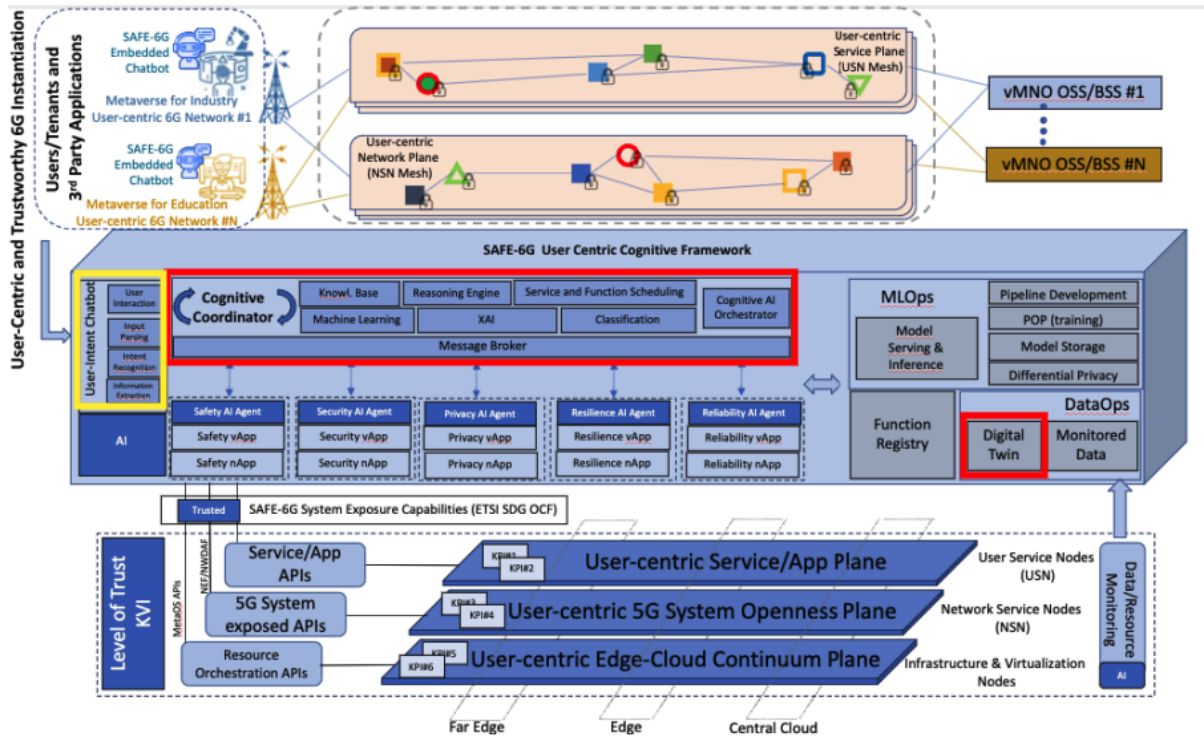


Figure 1: Functional View of SAFE-6G Architecture as presented in D2.2

CoCo therefore bridges the user-centric intent layer, represented by the Chatbot, with the mechanisms responsible for evaluating and enforcing trustworthiness within the SAFE-6G system. Building on the architectural definition established in D2.2 (Figure 1) and further elaborated in D4.1, the current implementation realizes an evolved CoCo architecture (Figure 2). CoCo maintains the required user and session context and governs the end-to-end process through improved component interoperability, a clearer distribution of responsibilities, the introduction of components such as the User Registry, and the central role of the Cognitive AI Orchestrator (CAIO). At a high level, CoCo integrates the following capabilities:

- The User Registry, Digital Twin, and Graphical User Interface provide and maintain information required for trustworthiness provisioning, while also supporting visualization and inspection of CoCo's internal decisions.
- The Machine Learning and XAI modules interpret user intents, predict trustworthiness requirements, support their translation into deployable actions, and provide transparency and explainability for the resulting outcomes.

- The Knowledge Base, Conflict Resolution Manager, and Service and Function Scheduling mechanisms identify and resolve conflicts and coordinate Trust Function deployments in a feasible, ordered, and conflict-free manner.
- The Message Broker enables asynchronous communication with the Trust Functions and conveys the final deployment decisions required to achieve the trustworthiness state requested by the user.
- The Cognitive AI Orchestrator governs the overall process and maintains the session and user information required throughout the lifecycle.

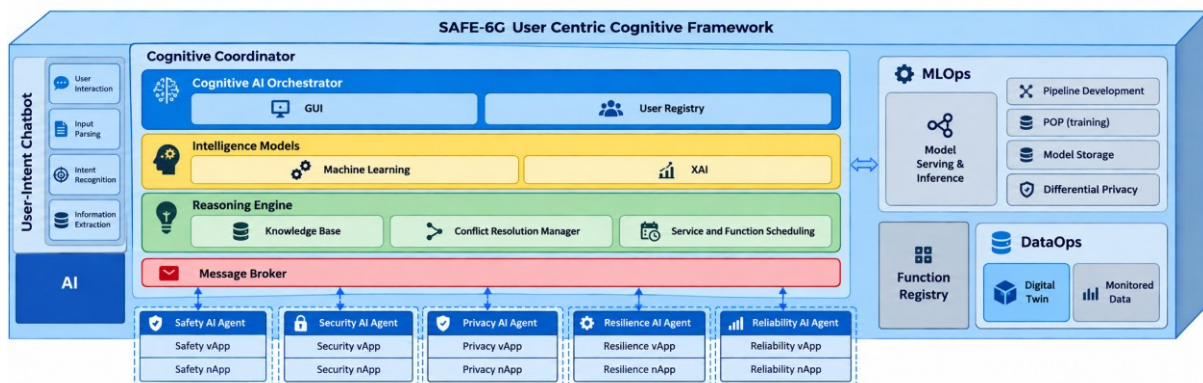


Figure 2: Evolved View of SAFE-6G User-Centric Cognitive Framework

The following subsections of Section 2 describe the architecture and functional design of these internal components according to the end-to-end processing flow of a user request. Particular attention is given to the responsibility of each component and to its contribution to transforming user-defined trustworthiness requirements into a coherent and actionable SAFE-6G configuration. Section 3 then presents the trustworthiness lifecycle as a sequential yet dynamic process, while Section 4 details the development and implementation decisions, including the principal technologies and libraries used. Although the implementation described in these sections is specific to SAFE-6G trustworthiness provisioning, Section 5.1 shows how the same architectural roles can be abstracted into a generic, profile-driven CoCo framework for intent-based network management.

2.1 INTERFACE WITH CHATBOT

The SAFE-6G Chatbot is the main user-facing component through which users express their trustworthiness requirements. As described in D2.2, the interaction relies on natural-language processing (NLP), allowing users to formulate their requirements as unstructured natural-language expressions. The Chatbot processes these expressions and performs intent detection and classification, associating the identified intents with the five SAFE-6G trustworthiness dimensions.

Architecturally, the Chatbot acts as an abstraction layer between the user and the trustworthiness-coordination mechanisms. Its NLP pipeline transforms the raw user interaction into the structured representation expected by CoCo, associating the relevant expressions with their corresponding Trust Function labels.

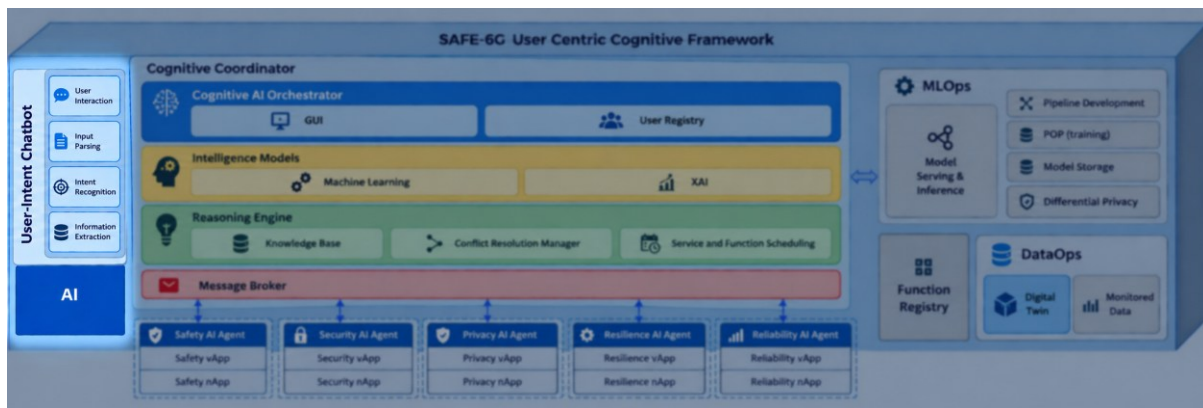


Figure 3: SAFE-6G Chatbot

Once the interaction has been completed, the structured intents, together with the required user-related information, are forwarded through the API exposed by CoCo. The interaction is bidirectional: after CoCo completes the coordination and trustworthiness-provisioning tasks, the outcome is returned to the Chatbot and communicated to the user. D4.1 introduced this interaction as a lightweight API-based exchange between the natural-language interface and the cognitive coordination pipeline; the implementation is described in detail in Section 4.1.

2.2 COGNITIVE AI ORCHESTRATOR

The Cognitive AI Orchestrator (CAIO) coordinates the execution of the CoCo trustworthiness-provision lifecycle and maintains the state information required throughout its different stages. It governs the progression of each user request, ensuring that the appropriate processing step is triggered and that the corresponding internal component is engaged.

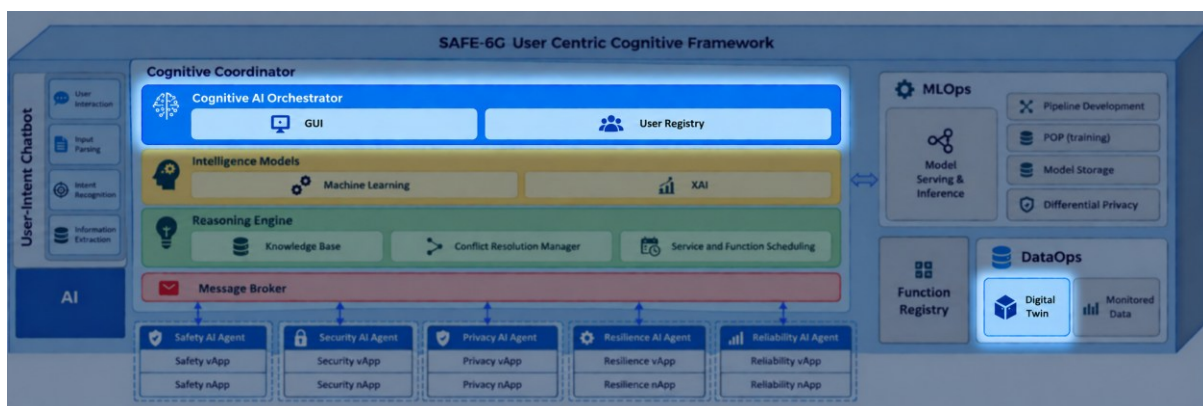


Figure 4: GUI, User Registry and Digital Twin's Involvement Constitute the Cognitive AI Orchestrator

In parallel, it exposes the maintained session information to the CoCo GUI for visualization. In this way, the CAIO connects the individual CoCo operations into a controlled end-to-end workflow, from the initial reception of a request to the production of the final outcome, while preserving the information required to inspect and trace each session.

To support this process, the orchestrator exchanges information with the following directly associated components:

- The CoCo Graphical User Interface (GUI), which provides a visualization and monitoring layer for the trustworthiness lifecycle. The CAIO exposes session-related APIs to the GUI, providing the required session and user information.
- The User Registry, which stores, updates, and retrieves user- and session-related information required for multi-user and multi-tenant operation throughout the lifecycle.
- The Digital Twin Prediction Module, which provides information about the system state and allows the CAIO to determine whether the workflow can proceed under the prevailing connectivity conditions.

2.2.1 CoCo GRAPHICAL USER INTERFACE

The CoCo GUI, is a newly introduced component to complement the overall SAFE-6G solution. It provides the visualization and monitoring layer of the CAIO, allowing the execution of user requests to be tracked throughout the trustworthiness provision lifecycle. It presents the information maintained by the CAIO, including the current lifecycle stage and the relevant session information produced as the request progresses through the different CoCo components.

The GUI operates as a passive monitoring component and does not participate in the orchestration or decision-making process. Instead, it periodically polls the CAIO to retrieve the latest available session information and updates its visualization accordingly. In this way, the GUI provides a near-real-time view of the CoCo workflow while remaining decoupled from the underlying execution logic.

2.2.2 USER REGISTRY

The User Registry is responsible for maintaining the user-related information required throughout the trustworthiness provision lifecycle. It acts as a database in which the information accompanying the initial request from the SAFE-6G Chatbot, such as the user identifiers, user role and other relevant user context, is stored or updated and associated with the corresponding user and session.

Maintaining this information centrally is important because several stages of the SAFE-6G lifecycle such as the TFs' AI agents as well as their deployment actions depend on the identity and context of the requesting user. The stored information allows the CAIO to preserve the appropriate user context across the complete processing flow, supports multi-user and multi-tenant operation and makes relevant subscriber information available when required by the Trust Functions. Additionally, the user role maintained in the registry is used to determine the trustworthiness levels accessible to the corresponding user, ensuring that subsequent trustworthiness decisions remain aligned with the user's assigned privileges.

2.2.3 DIGITAL TWIN PREDICTION MODULE

The Digital Twin Prediction Module provides the Cognitive AI Orchestrator (CAIO) with information about the radio conditions associated with a user before the trustworthiness provision workflow proceeds to its subsequent stages. Within the overall CoCo lifecycle, the module acts as an initial connectivity assessment mechanism, allowing the CAIO to determine whether the user is located in an area where the radio conditions are sufficient to support the requested service.

The motivation behind this functionality is to avoid initiating the complete trustworthiness provision workflow when the communication conditions of the user are already inadequate. Subsequent stages of CoCo involve several operations, including interpretation of the requested trustworthiness levels, reasoning over the available Trust Function configurations, conflict resolution, service and function scheduling, and communication with the Trust Functions. These operations are meaningful only when sufficient connectivity is available to support the requested service. Consequently, the radio-condition assessment is positioned at an early stage of the CoCo lifecycle and acts as a prerequisite for continuing the workflow.

To support this assessment, the module estimates the radio conditions at the user's location based on information originating from the Exata-based Digital Twin. In particular, the considered radio-quality indicators are Reference Signal Received Power (RSRP) and Signal-to-Interference-plus-Noise Ratio (SINR). These provide complementary information regarding the expected radio conditions, with RSRP characterizing the received reference-signal power and SINR reflecting signal quality relative to interference and noise.

Based on the estimated values, the CAIO determines whether the connectivity conditions are sufficient for the CoCo lifecycle to continue. The Digital Twin Prediction Module is therefore not responsible for the trustworthiness decision itself. Instead, it provides contextual network information that introduces spatial radio awareness into the CAIO orchestration process and supports an early connectivity-aware decision before the remaining trustworthiness coordination mechanisms are activated.

2.3 MACHINE LEARNING MODULE

The Machine Learning Module is the main inference component of CoCo and is responsible for translating the trustworthiness requirements expressed by the user (and received via the Chatbot's request) into quantitative trustworthiness values. As described in D4.1, the module processes the structured intents originating from the Chatbot and uses a Bidirectional Encoder Representations from Transformers (BERT) model to capture the contextual meaning of the user's expressions. Based on this representation, the module (trained on a domain-specific dataset) predicts the numerical level of trustworthiness associated with the relevant SAFE-6G dimensions: Safety, Security, Privacy, Resilience and Reliability.

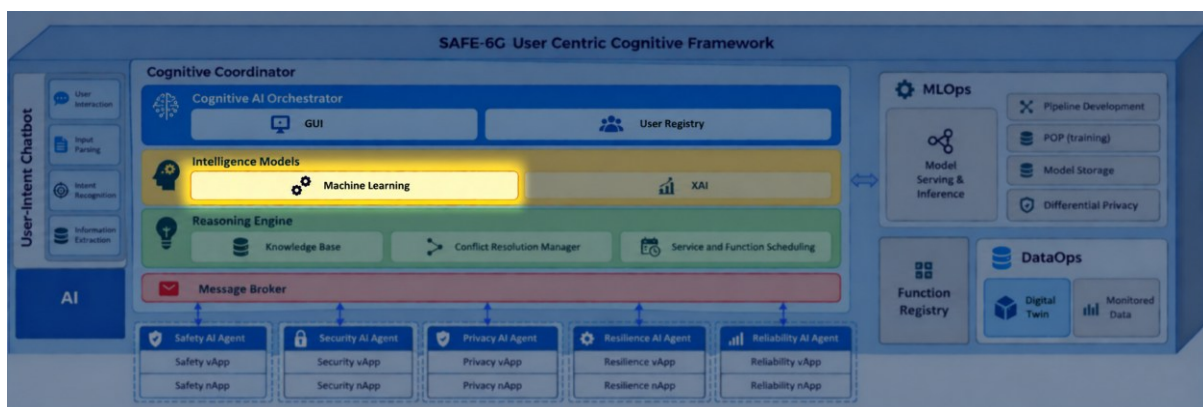


Figure 5: Machine Learning Module of CoCo

From an architectural perspective, the module follows a multi-dimensional regression approach, where initially BERT processes the received structured intents and provides a shared representation of them. This representation subsequently is routed to five dedicated regression heads one per Trust Function. Each intent is processed according to its associated Trust Function, producing a trustworthiness score in the range [0,100]. When multiple intents are received, their predictions are combined while considering the relative presence of each trustworthiness dimension in the overall user request, in a form of weighted normalization. The resulting values form the normalized Level of Trustworthiness (nLoTw), which represents the initial trustworthiness requirements across the five dimensions.

Within the CoCo lifecycle, the Machine Learning Module is the next processing step after the Digital Twin Prediction Module has given the request the green light to proceed. At this point, the network conditions have been considered and the focus shifts to determining what level of trustworthiness the user is requesting. The nLoTw produced by the module captures this request numerically and becomes the input for the following coordination stages, where the Trust Functions assess how these requested levels can be realized.

2.4 MESSAGE BROKER

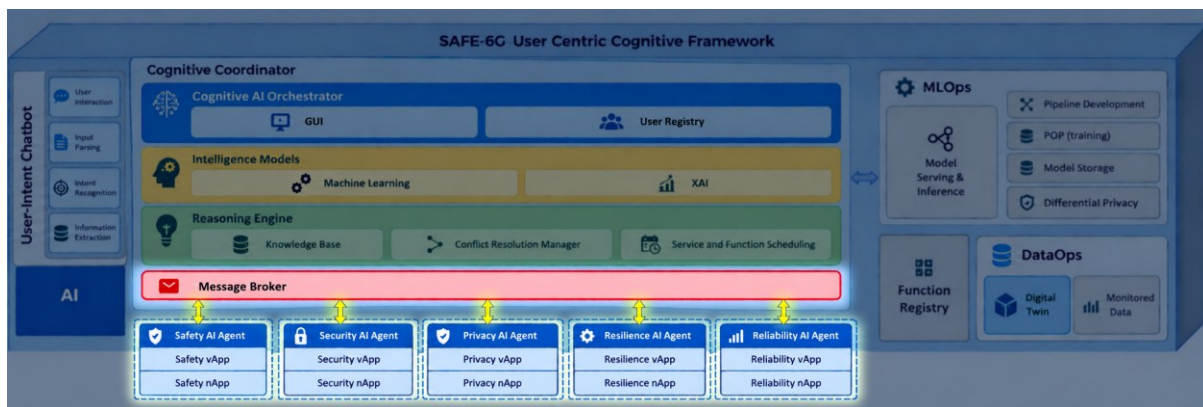


Figure 6: Message Broker Component and its Interaction with the Trust Functions

The Message Broker provides the communication layer between CoCo and the five Trust Functions, enabling the exchange of trustworthiness information without requiring direct communication between CoCo and each TF. As introduced in D4.1, this communication follows an asynchronous, message-based approach, supporting the decoupled operation of the TFs within the broader SAFE-6G architecture.

Within the lifecycle, the Message Broker becomes particularly relevant after the Machine Learning Module has produced the nLoTw. Before the information is forwarded, the required context for the request is gathered and the corresponding messages are prepared for the involved Trust Functions. These messages carry the requested trustworthiness information together with the user and session context required by each TF to evaluate the request. Each involved TF can then assess the requested level according to its own capabilities and operational conditions and return its feedback to CoCo through the same communication layer.

The interaction is therefore bidirectional. In the first direction, CoCo communicates the requested trustworthiness levels to the relevant TFs and collects their responses for the subsequent coordination and conflict-resolution stages. Later in the lifecycle, once the final cLoTw and deployment flavors have been determined, the Message Broker is used again to communicate the final configuration to the involved TFs. Their execution outcomes are subsequently returned to CoCo, allowing the lifecycle to continue only after the required responses from the participating Trust Functions have been collected.

2.5 REASONING ENGINE

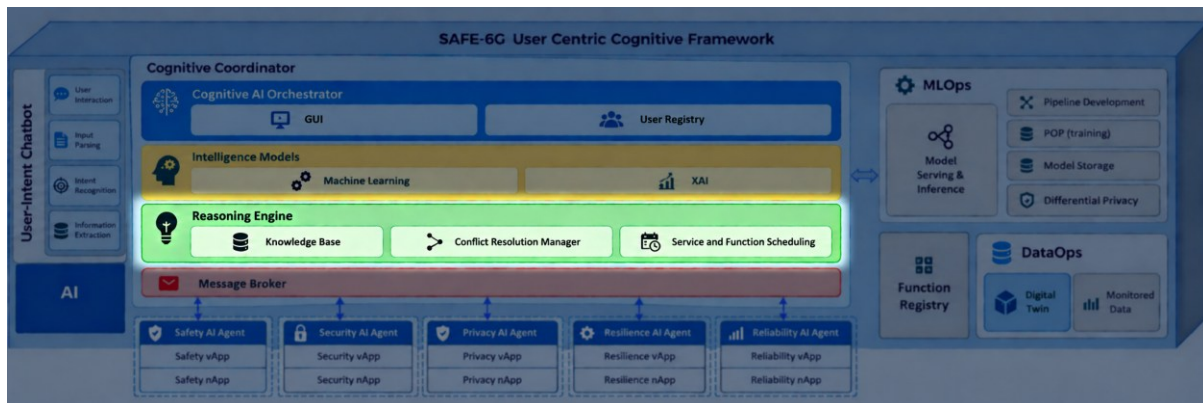


Figure 7: Reasoning Engine, Constituted by the Knowledge Base, the Conflict Resolution Manager and Service and Function Scheduling

The Reasoning Engine refines the trustworthiness configuration after the involved Trust Functions have evaluated the nLoTw received from CoCo and returned their proposed trustworthiness levels and deployment flavors. At this stage, the individual TF decisions must be considered jointly because configurations that are valid independently may become incompatible when applied together. The Reasoning Engine performs this cross-TF reasoning and produces a coherent configuration that can proceed to deployment.

Architecturally, the Reasoning Engine comprises two closely related elements: the Knowledge Base and the Conflict Resolution Manager (CRM), which also incorporates the Service and Function Scheduling capability.

2.5.1 KNOWLEDGE BASE

The Knowledge Base provides the Conflict Resolution Manager with the domain knowledge required to identify and resolve incompatibilities between Trust Function configurations. As anticipated in D2.2 and further elaborated in D4.1, the reasoning process relies on explicit knowledge describing the relationships and possible conflicts between the different trustworthiness dimensions when it comes to configuring SAFE-6G in a specific state. In the current architecture, this knowledge captures the combinations of TF flavors that cannot coexist and the rules that determine how such situations should be handled.

Keeping this knowledge separate from the reasoning process is important because the Reasoning Engine can evaluate the current configuration against a common and explicitly defined set of

constraints rather than embedding individual conflict cases directly into the coordination flow. The Knowledge Base can consequently evolve as additional cross-TF dependencies and conflict conditions are identified, without changing the overall role of the Reasoning Engine.

2.5.2 CONFLICT RESOLUTION MANAGER AND SERVICE FUNCTION SCHEDULING

The Conflict Resolution Manager (CRM), together with the Service and Function Scheduling capability, is engaged by the CAIO after the Message Broker has collected the required TF feedback. It identifies and resolves conflicts among the proposed TF deployments and derives the finalized, conflict-free deployment actions. To do so, it combines the trustworthiness levels proposed by the TFs with the conflict rules maintained in the Knowledge Base and evaluates the resulting configuration using its internal reasoning engine.

After conflict resolution, the component determines the scheduling of the Trust Functions required by the final configuration. Using the resolved configuration, the relationships among the involved functions, and the ordering information stored in the Knowledge Base, it determines the appropriate execution sequence when dependencies or interactions require one Trust Function to be deployed before another. The outcome consists of the conflict-free cLoTw scores, the corresponding deployment flavors, and the scheduling information required for the execution stage.

2.6 EXPLAINABLE AI

The XAI Aggregator improves the transparency of the Machine Learning models used within SAFE-6G and thereby supports trust in, and adoption of, AI-driven decisions. This is particularly relevant because the Trust Functions are managed by AI controllers that make decisions related to resource allocation, security, and other trustworthiness dimensions.

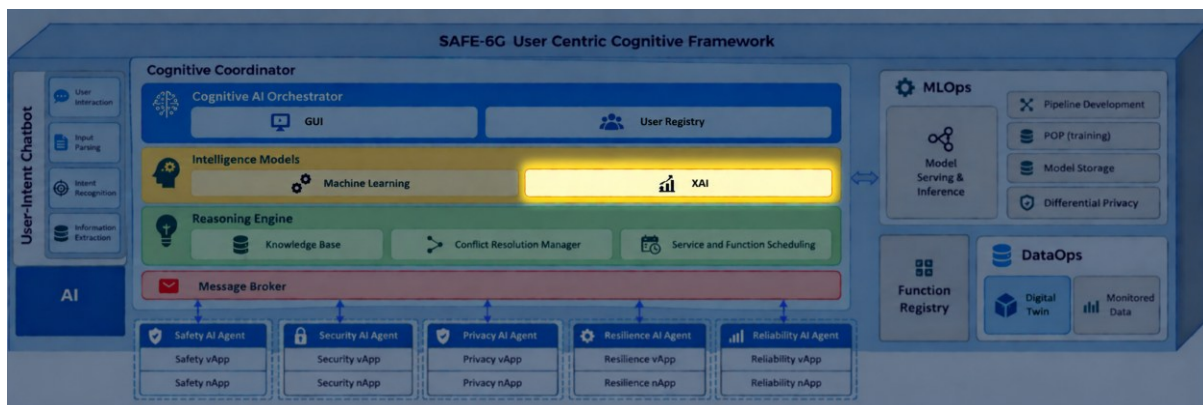


Figure 8: XAI Aggregator as part of CoCo

Operationally, the XAI Aggregator interacts with the user through the Chatbot, with CoCo acting as the intermediary. Once the user–Chatbot interaction is complete, the structured request is forwarded to CoCo, which orchestrates the trustworthiness-provision lifecycle. At the end of that lifecycle, CoCo automatically triggers the XAI Aggregator and supplies the required context, including the predictions to be explained and the Trust Functions that produced them.

The XAI Aggregator requires additional information to generate an explanation that is relevant to the user. This information falls into two categories, each addressing a different aspect of the explanation process.

The first category concerns the operational context in which the user query must be interpreted. An explanation is meaningful only when it is grounded in the platform that produced the underlying decision. For example, if a user asks why latency is being experienced, the module must know which Trust Function is responsible for the relevant decision and which input features that TF uses, such as memory state or network bandwidth. Without this contextual knowledge, a technically valid explanation could still fail to reflect the actual reasoning of the deployed model. The domain description therefore anchors each explanation in the concrete SAFE-6G environment and in the Trust Function that produced the prediction.

The second category of information concerns the explainability methods themselves. The XAI Aggregator relies on a catalogue of explainability methods, each of which is described by a structured entry capturing its applicability domain and its methodological assumptions. The catalog of the different methods is presented in deliverable D4.1. Each catalogue entry specifies the explanation type (local, global or both), the supported task types (classification, regression), the input feature structure (tabular, text), whether the model must be differentiable, the compatible model architectures, and the methodological assumptions underlying the interpretation of the method. Each method in the catalogue relies on methodological assumptions (such as feature independence for Permutation Importance or a representative background dataset for Kernel SHAP) and has known limitations in terms of the model architectures, the data types and the task types it supports. The XAI assistant must be aware of these assumptions and limitations to avoid over-interpreting the results: an attribution value is not a universal measure of importance, but a quantity whose meaning depends on the method that produced it and on the conditions under which that method remains valid. This information is exploited by the interpreter to verify, for each explanation, whether the assumptions of the selected method are satisfied, and to warn the user with a caution whenever an assumption is at risk of being violated.

Together, this information allows the LLM-based XAI logic to identify the Trust Functions involved in a user flow and the input features for which an explanation is required. The domain description helps the LLM recognize platform-specific vocabulary and route the explanation to the relevant Trust Function, while the method descriptions and assumptions support reasoning about the reliability and limitations of the generated explanation.

3 COGNITIVE COORDINATOR LIFECYCLE

The interaction among the CoCo components follows a well-defined sequence while remaining dynamic under the governance of the CAIO, which reacts to lifecycle events and completion conditions (for example, the reception of all expected Trust Function feedback messages).

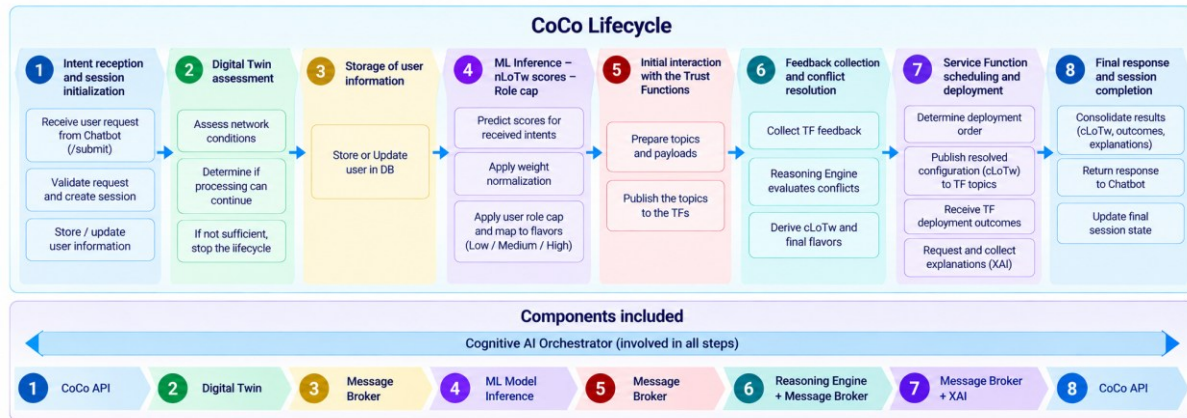


Figure 9: CoCo's Lifecycle View

Figure 9 illustrates the resulting end-to-end sequence and identifies the components involved at each stage:

- 1. Intent reception and session initialization.** The lifecycle begins when the SAFE-6G Chatbot forwards a structured user request through the /submit API exposed by CoCo. The payload contains the trustworthiness intents together with the relevant user information. After validating the request, the CAIO creates a new processing session, associates it with the corresponding user, maintains its evolving state throughout the lifecycle, and exposes the relevant information to the GUI through dedicated APIs.
- 2. Digital Twin assessment.** After session initialization, the CAIO invokes the Digital Twin Prediction Module to assess the network conditions associated with the requesting user. The result acts as an early decision point: when the predicted connectivity conditions are sufficient, the workflow proceeds; otherwise, the lifecycle is terminated before trustworthiness inference and coordination are initiated.
- 3. Storage of user information.** Following a positive Digital Twin assessment, the user information accompanying the initial Chatbot request is stored or updated in the User Registry. This makes identifiers, organizational role, and other contextual information available to the subsequent lifecycle stages.
- 4. ML inference, nLoTw scores, and role cap.** The received intents are processed by the Machine Learning Module, which predicts an individual trustworthiness score for each intent according to its associated trustworthiness dimension. Intent-based weighting and normalization consolidate these predictions into the five-dimensional nLoTw. Before the values leave this stage, the CAIO applies the role-based trustworthiness cap maintained in the User Registry. The resulting scores are then mapped to the Low, Medium, or High flavors used in the interaction with the Trust Functions.

5. **Initial interaction with the Trust Functions.** The Message Broker prepares the topics and payloads required by the involved Trust Functions. These payloads combine the nLoTw scores and flavors with the user, session, and contextual information required by each TF. CoCo then publishes the messages through dedicated Kafka topics, allowing every involved TF to evaluate the requested trustworthiness configuration independently.
6. **Feedback collection and conflict resolution.** The feedback produced by the involved Trust Functions is received through the Message Broker and associated with the corresponding session. Once all expected feedback has been collected, the CAIO triggers the Reasoning Engine to evaluate the proposed scores and flavors for cross-TF conflicts. When incompatibilities are identified, the CRM applies the corresponding reasoning logic and iteratively evaluates the updated configuration until a stable result is obtained. The outcome is the cLoTw together with the final TF flavor mappings to be deployed.
7. **Service Function scheduling and deployment.** Based on the resolved configuration, the Reasoning Engine invokes its Service and Function Scheduling capability, implemented as part of the CRM, to determine the appropriate execution order of the involved Trust Functions when dependencies require sequencing. The resulting cLoTw scores and final flavors are published through the Message Broker to the corresponding TFs. CoCo then collects their deployment outcomes and, once the required responses have been received, requests the corresponding explanations from the XAI module.
8. **Final response and session completion.** After the deployment and explainability stages have concluded, CoCo consolidates the cLoTw, deployment outcomes, and generated explanations into the final session result. This information is returned to the SAFE-6G Chatbot through the CoCo API, allowing the outcome of the trustworthiness-provision lifecycle to be communicated to the user. The session state is then updated to reflect completion.

4 COGNITIVE COORDINATOR – IMPLEMENTATION AND DEVELOPMENT DECISIONS

4.1 INTERFACE WITH CHATBOT

The entry point of CoCo is the API interface used to receive user intents from the SAFE-6G Chatbot. While the Chatbot is not a direct subcomponent of CoCo, it is important to underline its importance as well as the interface between them. Starting, CoCo exposes a dedicated **/submit** endpoint, as implemented in **data_controller.py**, through which the Chatbot forwards the collected intents, together with the user-related information required during the subsequent processing stages. The exchanged information follows a predefined JSON structure, as presented in Figure 10 providing CoCo with the input required to initiate a new trustworthiness provision flow.

```

{
  "name": "Hendrix",
  "msisdn": "300000000001",
  "supi": "imsi-001003055583003",
  "role": "Director",
  "Hardware_id": "android-001",
  "data": [
    {
      "label": "Reliability",
      "text": "I need a reliable connection"
    },
    {
      "label": "Security",
      "text": "I need to be secure from attacks"
    }
  ]
}

```

Figure 10: Chatbot Payload Towards CoCo

The API is developed using the **FastAPI** [1] framework, while the received payload is validated using **Pydantic** [2] to ensure that the expected structure and all required fields are provided before further processing. Following successful validation, the CAIO initializes the corresponding processing session and prepares the received data for use by its internal components. The intents are organized according to their associated Trust Function labels, while the accompanying user information is made available to the components that require it during the lifecycle.

The HTTP request remains open throughout the complete CoCo processing flow. Rather than returning an intermediate result immediately after the intents have been received, the request is completed once the subsequent coordination stages (publishing of trustworthiness scores to the TFs, conflict resolution, deployment completion, explainability provided) have concluded. CoCo then returns the consolidated outcome through the same interface, allowing the Chatbot to communicate the result of the trustworthiness provision process back to the user as illustrated in Figure 11. The complete list with the variables exchanged through the interface is presented in Table 2.

```
"deployment_complete": true,
"privacy": "80.0",
"reliability": "20.0",
"resilience": "0.0",
"security": "0.0",
"safety": "0.0",
"explainability": "Based on your request, your role in your organization and the observed limitations..."
```

Figure 11: CoCo's Response to the Chatbot after the Session is Finalized

Variable	Type	Origin	Description/value
name	string	Chatbot	The name of the user accessing the chatbot
msisdn	string	Chatbot	The MSISDN provided by the user
supi	string	Chatbot	The user's SUPI as identified in Chatbot's preceding steps
role	string	Chatbot	The user's role: [Technician, Line Manager, Instructor, Director]
data	list	Chatbot	List containing the user intents
label	string	Chatbot	The TF-mapped label of the user's intents
text	string	Chatbot	User's intent as identified by the Chatbot
deployment_complete	boolean	CoCo	True or False. The outcome of the trustworthiness deployments over the TFs
<tf_name>	string	CoCo	The provisioned trustworthiness score per TF
explainability	string	CoCo	The explainability over the performed actions as CoCo received it from the xAI module

Table 2: Exchanged variables via the CoCo API

4.2 COGNITIVE AI ORCHESTRATOR

As described in the architectural section, the CAIO coordinates the trustworthiness-provision lifecycle. It determines when a processing stage has been completed, whether the conditions required to continue have been satisfied, and which stage should be executed next. To support this role, the CAIO combines session-management logic with information provided by the User Registry and the Digital Twin Prediction Module. In parallel, it exposes the information required by the GUI to provide visibility into, and enable inspection of, the trustworthiness lifecycle.

The core orchestration functionality is implemented through the ***SessionOrchestrator*** class in ***session_orchestrator.py*** as an in-process Python singleton. Each request is represented through a ***SessionState*** dataclass and maintained in memory throughout its execution. The session progressively collects the information produced during the lifecycle, including *user metadata and intents, predicted trustworthiness scores, Trust Function feedback, resolved cLoTw values, deployment outcomes, lifecycle state and event logs*.

The User Registry and Digital Twin Prediction Module provide two important sources of information for this orchestration. Information maintained in the User Registry is associated with the corresponding user, allowing the CAIO to preserve the user context throughout the lifecycle and make it available where required, including for the information exposed to the CoCo GUI.

An important CAIO function is the application of the user-role cap. The relevant role is retrieved from the User Registry immediately after the Machine Learning Module produces the trustworthiness predictions. The CAIO then applies the corresponding maximum trustworthiness allowance before the nLoTw scores proceed to the subsequent coordination stages. In the current configuration, the maximum score is 39.99 for Technicians, 79.99 for Line Managers, and 100 for Instructors and Directors. Values exceeding the applicable limit are capped accordingly, ensuring that the trustworthiness configuration communicated to the Trust Functions remains within the privileges assigned to the requesting user. For example, a predicted Reliability score of 80.2 for a Technician is reduced to the maximum value permitted for that role.

The Digital Twin Prediction Module provides an outcome that directly affects session progression. Based on this assessment, the CAIO determines whether the request can proceed to the subsequent processing stages or whether the flow should terminate. In the current implementation, the CAIO compares the predicted RSRP and SINR values with predefined thresholds to ensure that signal quality is sufficient for the user session. Both Digital Twin outputs must exceed their respective thresholds before the lifecycle proceeds. Otherwise, the CAIO terminates the session and returns an appropriate response to the initial /submit request, allowing the Chatbot to inform the user.

A similar event-driven approach is used for interaction with the Trust Functions. The CAIO is connected to the Kafka producers and consumers of the Message Broker and tracks the messages published and received throughout the lifecycle. Reception of the expected TF feedback or deployment outcomes is reflected in the corresponding session state and determines when the next processing step can be triggered. For example, conflict resolution begins only after feedback from all involved TFs has been received. In this way, the CAIO acts as the lifecycle controller, transforming the asynchronous events exchanged between CoCo and the TFs into an ordered sequence of processing stages.

Finally, the CAIO makes the maintained runtime state available to the CoCo GUI through the **FastAPI** interface defined in **data_controller.py**. The exposed API allows the GUI to periodically retrieve session and processing information, including the current lifecycle state and the intermediate and final information accumulated during execution. The GUI can therefore track the evolution of each request using the same session state that drives the orchestration logic, without directly participating in the lifecycle execution. The endpoints that the CAIO exposes to the CoCo GUI are provided in the following section to complement the GUI's description.

Table 3 summarizes the information maintained by the CAIO throughout the lifecycle, regardless of whether it is generated by the CAIO itself or obtained from another component, such as the User Registry or Message Broker. The same information is made available to the GUI for monitoring and inspection.

Variable	Description
name	User's/Tenant's name
msisdn	User's/Tenant's MSISDN
supi	User's/Tenant's SUPI
role	User's/Tenant's role

hardware_id	User's/Tenant's device identifier.
session_id	Unique identifier of a CoCo request/session, generated by CAIO. Used to track one interaction through the full workflow.
status	Current high-level state of the session, such as submitted, collecting feedback, conflict resolved, completed or failed.
lifecycle_state	Step-by-step progress map of the workflow stages, used by the GUI lifecycle view.
events	Chronological log messages describing what happened during the session.
data	Submitted classified intents/text data received from the Chatbot and used for nLoTW prediction.
role_cap_changes	Records which predicted scores were reduced because of the user-role cap.
predicted_scores	Raw nLoTW scores predicted by the ML model before role-based capping.
predicted_flavors	Flavor labels derived from the predicted scores, such as low, medium or high.
Tf_feedback_scores	Scores returned by the Trust Functions after receiving the initial nLoTW values.
resolved_scores	Final scores after conflict resolution, also referred to as cLoTW values.
resolved_flavors	Final flavor labels after conflict resolution.
deployment_queue	Ordered list of Trust Functions that must perform deployment actions.
outcomes	Deployment results returned by the Trust Functions. OK or NOK.
final_response	Final response prepared by CoCo for the Chatbot, including completion status, final trustworthiness values and explainability outcome.

Table 3: Session information maintained by the CAIO

4.2.1 CoCo GRAPHICAL USER INTERFACE

The CoCo GUI, whose source code is located in the gui directory of the provided repository, is implemented as a web-based monitoring interface using React and plain CSS. The frontend is separated from the CoCo processing logic and acts exclusively as a visualization layer over the runtime state exposed by the CAIO. It does not communicate directly with the AI Model, Message Broker, Reasoning Engine, or Trust Functions. Instead, it periodically polls the CAIO API and uses the returned session data to update the displayed workflow state.

The GUI is implemented as a single-page React application. API communication is centralized in a frontend client module that wraps the requests used by the interface. The main application component maintains the list of available sessions, the currently selected session, backend status information, and polling errors as React state. Three polling loops are used: GET /data/sessions retrieves the available CoCo sessions every second; GET /data/session/{session_id} retrieves the latest full snapshot for the selected session every second; and GET /data/status retrieves lightweight backend status information, including whether processing is active and how many data items are stored. This polling-based design keeps the GUI passive with respect to the CoCo workflow. It does not trigger individual processing stages or make orchestration decisions; instead, it observes the latest backend state and re-renders the interface when that state changes. Consequently, the GUI can follow a running session in near real time while remaining decoupled from the CAIO, Trust Functions, and deployment logic. The CAIO APIs used by the GUI are summarized in Table 4.

GUI call	Backend endpoint	Description
<i>api.getSessions()</i>	GET /data/sessions	Polls every second. Returns the session list in newest-first order for the sidebar: session_id, user_id, name, role, status, deployment_complete.
<i>api.getSession(sessionId)</i>	GET /data/session/{session_id}	Polls every second for the selected session. Returns the full session snapshot used by the dashboard: user/session metadata, lifecycle state, predicted/initial/resolved scores, TF feedback, deployment queue/current TF, outcomes, final response, and event log.
<i>api.getStatus()</i>	GET /data/status	Polls every second for backend status. Returns { calculating, data_count }, used by the GUI's header/status display.

Table 4: CAIO APIs exposed to the GUI

The session list returned by `/data/sessions` allows available CoCo sessions to be inspected and enables the GUI user to select a previous or active session to inspect. When automatic tracking is enabled, the GUI follows the most recent session in the list. If the GUI user manually selects a session, the interface stops following the latest session and displays the selected one instead.

The full session snapshot returned by `/data/session/{session_id}` is the primary source of information for the interface. It includes session and user metadata, submitted intents, lifecycle state, predicted nLoTw scores, role-cap information, Trust Function feedback, resolved cLoTw values, deployment outcomes, final response data, and orchestration events. The frontend treats this snapshot as the authoritative representation of the workflow and does not reproduce the backend orchestration logic locally. Instead, it derives its visual elements from the fields already maintained by the CAIO. Figure 12 illustrates the range of information contained in the snapshot, including user and session identifiers, predicted and resolved scores, and logged events.

<pre> </> Session Snapshot { "session_id": "e2e-eucnc-4d355f59-8617-4f13-9a45-c57497c7c76e", "user_id": "81", "name": "e2e-user", "email": "1000000001", "supi": "001011000000011", "role": "instructor", "UC": "UC1", "Hardware_id": "e2e-hardware", "tbd": null, "status": "COMPLETED", "intents": [], "role_cap": null, "role_cap_changes": {}, "predicted_scores": {}, "predicted_flavors": {}, "expected_feedback_tfs": ["Resilience"], "resolved_result": { "user_id": "81", "cLoTw": { "Privacy": 0, "Security": 0, "Reliability": 0, "Resilience": 98.85, "Safety": 0 }, "resolved_scores": { "Privacy": 53.64, "Security": 53.64, "Reliability": 53.64, "Resilience": 98.85, "Safety": 53.64 }, "resolved_flavors": { "Privacy": null, "Security": null, "Reliability": null, "Resilience": "high", "Safety": null } }, "lifecycle_state": { "submitted": "done", "role_cap": "skipped", "initial_scores": "done", "initial_published": "done", "tf_feedback": "done", "conflict_resolution": "done", "resolved_published": "done", "deployment_outcomes": "done", "deployment_complete": "done", "chatbot_response": "done" }, "events": ["Session created", "Predicted nLoTw score snapshot stored for GUI: {'Privacy': 53.64, 'Security': 53.64, 'Reliability': 53.64, 'Resilience': 98.85, 'Safety': 53.64}", "Role cap applied: cap=100.0, changes=none", "Active Trust Functions selected: ['Privacy', 'Security']", "Initial score snapshot stored for GUI: {'Privacy': 53.6442, 'Security': 53.6442, 'Reliability': 53.6442, 'Resilience': 98.85, 'Safety': 53.6442}", "Initial score stored for Privacy: 53.6442"] } </pre>	<pre> </> Session Snapshot { "session_id": "e2e-eucnc-4d355f59-8617-4f13-9a45-c57497c7c76e", "user_id": "81", "name": "e2e-user", "email": "1000000001", "supi": "001011000000011", "role": "instructor", "UC": "UC1", "Hardware_id": "e2e-hardware", "tbd": null, "status": "COMPLETED", "intents": [], "role_cap": null, "role_cap_changes": {}, "predicted_scores": {}, "predicted_flavors": {}, "expected_feedback_tfs": ["Resilience"], "resolved_result": { "user_id": "81", "cLoTw": { "Privacy": 0, "Security": 0, "Reliability": 0, "Resilience": 98.85, "Safety": 0 }, "resolved_scores": { "Privacy": 53.64, "Security": 53.64, "Reliability": 53.64, "Resilience": 98.85, "Safety": 53.64 }, "resolved_flavors": { "Privacy": null, "Security": null, "Reliability": null, "Resilience": "high", "Safety": null } }, "lifecycle_state": { "submitted": "done", "role_cap": "skipped", "initial_scores": "done", "initial_published": "done", "tf_feedback": "done", "conflict_resolution": "done", "resolved_published": "done", "deployment_outcomes": "done", "deployment_complete": "done", "chatbot_response": "done" }, "events": ["Session created", "Predicted nLoTw score snapshot stored for GUI: {'Privacy': 53.64, 'Security': 53.64, 'Reliability': 53.64, 'Resilience': 98.85, 'Safety': 53.64}", "Role cap applied: cap=100.0, changes=none", "Active Trust Functions selected: ['Privacy', 'Security']", "Initial score snapshot stored for GUI: {'Privacy': 53.6442, 'Security': 53.6442, 'Reliability': 53.6442, 'Resilience': 98.85, 'Safety': 53.6442}", "Initial score stored for Privacy: 53.6442"] } </pre>	<pre> </> Session Snapshot { "session_id": "e2e-eucnc-4d355f59-8617-4f13-9a45-c57497c7c76e", "user_id": "81", "name": "e2e-user", "email": "1000000001", "supi": "001011000000011", "role": "instructor", "UC": "UC1", "Hardware_id": "e2e-hardware", "tbd": null, "status": "COMPLETED", "intents": [], "role_cap": null, "role_cap_changes": {}, "predicted_scores": {}, "predicted_flavors": {}, "expected_feedback_tfs": ["Resilience"], "resolved_result": { "user_id": "81", "cLoTw": { "Privacy": 0, "Security": 0, "Reliability": 0, "Resilience": 98.85, "Safety": 0 }, "resolved_scores": { "Privacy": 53.64, "Security": 53.64, "Reliability": 53.64, "Resilience": 98.85, "Safety": 53.64 }, "resolved_flavors": { "Privacy": null, "Security": null, "Reliability": null, "Resilience": "high", "Safety": null } }, "lifecycle_state": { "submitted": "done", "role_cap": "skipped", "initial_scores": "done", "initial_published": "done", "tf_feedback": "done", "conflict_resolution": "done", "resolved_published": "done", "deployment_outcomes": "done", "deployment_complete": "done", "chatbot_response": "done" }, "events": ["Session created", "Predicted nLoTw score snapshot stored for GUI: {'Privacy': 53.64, 'Security': 53.64, 'Reliability': 53.64, 'Resilience': 98.85, 'Safety': 53.64}", "Role cap applied: cap=100.0, changes=none", "Active Trust Functions selected: ['Privacy', 'Security']", "Initial score snapshot stored for GUI: {'Privacy': 53.6442, 'Security': 53.6442, 'Reliability': 53.6442, 'Resilience': 98.85, 'Safety': 53.6442}", "Initial score stored for Privacy: 53.6442"] } </pre>
---	---	---

Figure 12: Information Stored in a Snapshot, consumed by the GUI

The interface itself, illustrated in Figure 13, is organized around the information maintained for the selected session. A session panel on the left allows the available CoCo sessions to be inspected and displays the corresponding user intents together with a radar representation of the nLoTw and cLoTw values across the five Trust Functions. The upper part of the main view presents the principal session and user information, including identifiers, user role, use case, current processing status and deployment-completion state. The backend status obtained from `/data/status` is used to indicate whether the API is actively processing data.

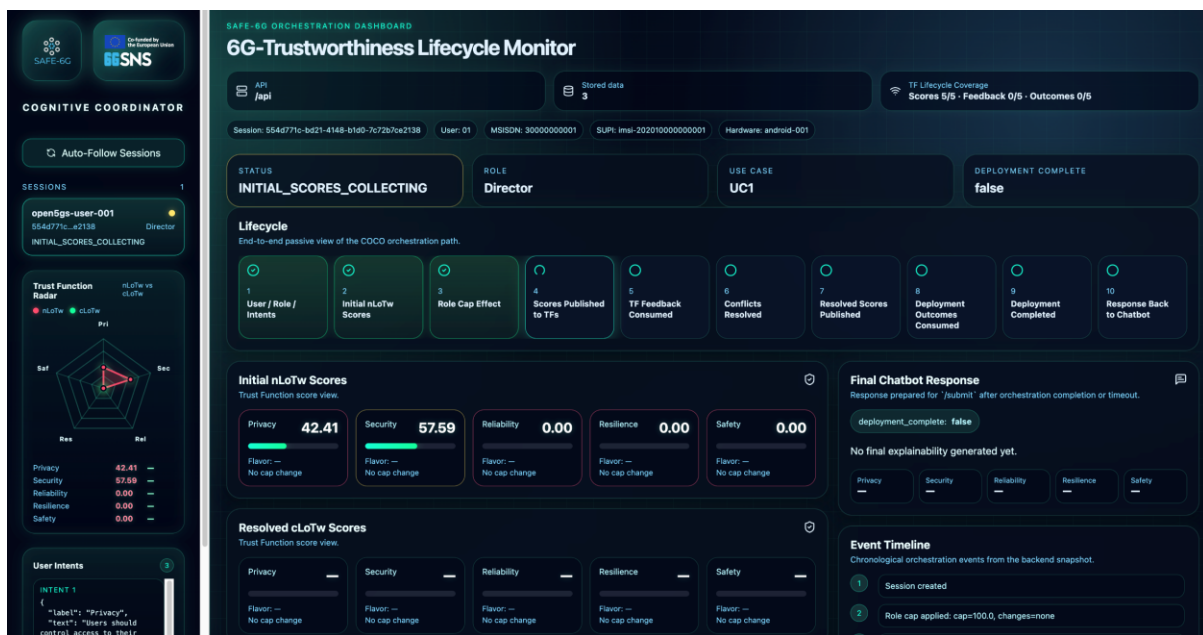


Figure 13: CoCo's Graphical User interface Overview

The central Lifecycle view, shown in Figure 14, visualizes the sequential progression of a session through the CoCo workflow. Each stage is represented individually, covering reception of user information and intents, generation of the initial nLoTw scores, application of the role cap, interaction with the Trust Functions, conflict resolution, publication of the resolved scores, collection of deployment outcomes, and preparation of the final response to the Chatbot. The current stage is derived from the lifecycle_state, status, and related progress fields in the latest session snapshot, allowing completed, active, and pending stages to be distinguished visually.

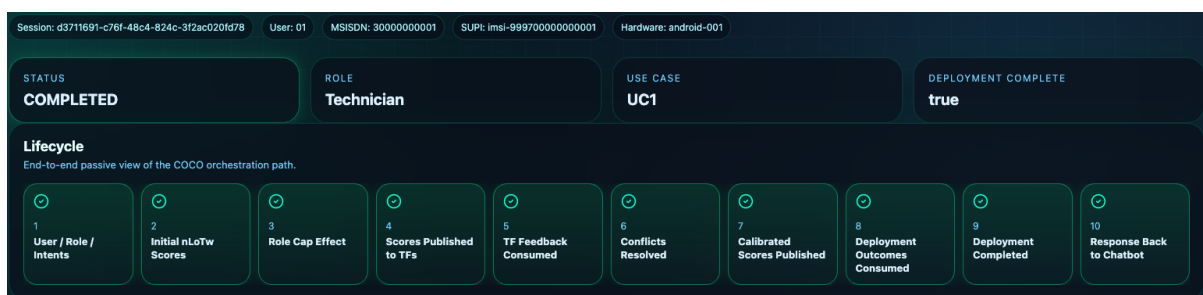


Figure 14: GUI's Central Lifecycle View

Additional panels expose the numerical and structured information produced during processing. The Initial nLoTw Scores panel presents the initial predicted score for each Trust Function together with its associated information, while the Resolved cLoTw Scores panel presents the corresponding values after conflict resolution. Deployment-related panels show the current deployment queue, resolved-score publication progress, and collected outcomes. The interface also includes the final response prepared for the Chatbot and an Event Timeline that presents the orchestration events recorded for the session in chronological order.

The GUI is decomposed into dedicated React components that correspond to the main visualization areas of the dashboard. The sidebar component manages the session list, session selection, displayed intents and radar-style score overview. The header component presents high-level session and backend information. The lifecycle component converts the current session state into a visual sequence of completed, active and pending workflow stages. Score-related components render the predicted nLoTw values, role-capped values and resolved cLoTw values for the five Trust Functions. Deployment status components present Trust Function feedback, resolved-score publication progress, deployment queues and collected outcomes. Separate components are also used for the final response, chronological event timeline and JSON inspection panels.

This decomposition keeps the GUI aligned with the structure of the session snapshot. Each component receives only the subset of session data that it needs, such as score fields, lifecycle fields, event lists or deployment outcome fields. When a new polling response is received, the main application state is updated and *React* re-renders the affected components. This allows the interface to update progressively during a live CoCo execution: lifecycle stages advance, score panels appear when values become available, deployment progress is updated as Trust Functions respond, and the final response is displayed once the session reaches completion.

The GUI is introduced to make the internal operation of CoCo observable during execution. Since the CoCo workflow involves several asynchronous components and intermediate processing stages, a polling-based monitoring interface provides a simple and reliable way to inspect the current state without interfering with the orchestration logic. By relying only on the state exposed by the CAIO, the GUI preserves separation between monitoring and processing, while still providing visibility into score generation, Trust Function interaction, conflict resolution, deployment progress and final response preparation. This makes the interface useful both for technical validation of the end-to-end workflow and for understanding how each user request evolves through the CoCo pipeline.

4.2.2 USER REGISTRY

The User Registry is implemented as a PostgreSQL database within CoCo and provides persistent storage for the user-specific information required during the trustworthiness-provision lifecycle. Together with the classified intents, the initial request received from the SAFE-6G Chatbot contains information such as the user name, MSISDN, SUPI, applicable use case (e.g., SAFE-6G multi-use-case demonstrations), hardware identifier (e.g., XR glasses), and user role. The roles currently supported by SAFE-6G are Technician, Line Manager, Instructor, and Director. Upon reception, this information is associated with a uniquely generated userID and stored in the User Registry. For new users, CoCo also assigns the corresponding user identifier, which is reused as a multi-tenancy helper in subsequent requests. Figure 15 illustrates the stored information.

user_id	name	msisdn	supi	role	hardware_id	created_at	updated_at
03	charlie	30000000003	imsi-202010000000003	Line Manager	android-003	2026-05-10 11:45:14.499988+00	2026-05-10 11:45:14.499988+00
04	diana	30000000004	imsi-202010000000004	Technician	android-004	2026-05-10 11:45:14.499988+00	2026-05-10 11:45:14.499988+00
50f2cc9ecb9d	hendrix	69669999999	imsi-202010999999999	Director	as7c7ea5cv	2026-05-10 11:47:11.419877+00	2026-05-10 11:47:11.419877+00
02	bob	30000000002	imsi-202010000000002	Director	as7c7ea5cv	2026-05-10 11:45:14.499988+00	2026-05-12 12:05:09.944189+00
01	open5gs-user-001	30000000001	imsi-202010000000001	Technician	android-001	2026-05-10 11:45:14.499988+00	2026-06-01 08:18:11.063253+00

Figure 15: Users entries stored in the User Registry

In implementation terms, the User Registry acts as the persistent identity layer behind the transient CoCo session state. When a new request is submitted by the Chatbot and forwarded by the CAIO session orchestrator, the registry checks whether an entry already exists for the provided SUPI. If an entry exists, the same userID is reused, while mutable information such as name, role, use case, hardware identifier, and additional metadata can be updated. If no entry exists, a new registry record is created, a unique userID is generated, and the complete user context is stored. This prevents duplicate records for repeated requests from the same subscriber. The user's SUPI is used as the database primary key. Creation and update of User Registry entries are handled by `user_registry_handler.py`. Table 5 summarizes the registry columns and their role within CoCo and the broader SAFE-6G framework.

Column	Datatype	Description/involvement
user_id	string	Uniquely generated identifier by the <code>user_registry_handler.py</code> that assists CAIO in multi-tenancy handling.
name	string	User's name. Provided through the <code>/submit</code> request by the Chatbot
msisdn	string	User's MSISDN. Provided through the <code>/submit</code> request by the Chatbot
supi	string	User's SUPI. Provided through the <code>/submit</code> request by the Chatbot. Communicated to the TFs via the Message Broker as it is important to their actions performed in the 5G Core.
role	string	User's role that defines the privileges (performed by CAIO's role capping) in the trustworthiness provision. Provided through the <code>/submit</code> request by the Chatbot
hardware_id	string	Unique identifier of the user's device through which he/she is accessing the SAFE-6G. Provided through the <code>/submit</code> request by the Chatbot. Communicated to the TFs via the Message Broker as it is important to their actions performed in the 5G Core.
UC	string, can be null	Use case identifier for possible future demonstration uses of CoCo. Provided through the <code>/submit</code> request by the Chatbot
created_at	timestamp with timezone	Creation timestamp of entry in the <code>user_registry</code> .
updated_at	timestamp with timezone	Last update timestamp of entry in the <code>user_registry</code> .

Table 5: User Registry Database Schema and Column Descriptions

4.2.3 DIGITAL TWIN PREDICTION MODULE

The detailed development and evaluation of the AI models supporting the Digital Twin are addressed within WP5, where the corresponding Digital Twin development and experimentation activities are formally undertaken. This section therefore focuses on the implementation of the prediction mechanism and, in particular, on how the resulting models are exploited by CoCo during its operational lifecycle.

The prediction capability is constructed using radio-environment information provided by the Exata-based Digital Twin. The Digital Twin generates data associating spatial locations with corresponding radio-quality measurements or estimates. Conceptually, each observation can be represented by a

spatial coordinate x and its corresponding radio-quality value r . Depending on the considered deployment, the spatial coordinate can represent a two-dimensional position, $x = (x, y)$, or can additionally incorporate receiver height through a three-dimensional representation, $x = (x, y, z)$. The associated radio information includes the RSRP and SINR values observed at these locations.

Rather than directly storing and querying these observations as a conventional discrete radio map, the prediction mechanism uses them to construct Implicit Neural Representations (INRs) [3]. An INR represents a signal or physical field through a coordinate-based neural network. In the considered implementation, the network receives a spatial coordinate and learns the corresponding value of the underlying radio field. The prediction problem can therefore be expressed as $f_{\theta}(x) \rightarrow r$, where x represents the spatial position, r represents the corresponding radio-quality metric, and θ denotes the trainable parameters of the neural network.

An important characteristic of this representation is its continuous formulation. Although the original Digital Twin data are available at a finite set of spatial locations, the trained INR learns a continuous mapping between spatial coordinates and radio-quality values. Consequently, the resulting model can also be queried at locations that were not explicitly contained in the original dataset. This is particularly relevant for CoCo, since the location of a user initiating a request does not necessarily coincide with one of the sampling locations used by the Digital Twin.

For the Digital Twin Prediction Module, the INRs are instantiated using Sinusoidal Representation Networks (SIRENs) [4] and implemented using PyTorch. SIRENs are coordinate-based neural networks employing sinusoidal activation functions in their hidden layers. Their ability to represent continuous signals with relatively complex spatial variations makes them suitable for learning radio fields, where signal conditions can vary across space due to propagation effects such as multipath, shadowing, and blockage.

The implementation can be divided into two logically separated phases: model preparation and operational inference. During model preparation, the radio dataset generated by the Exata Digital Twin is used to train the SIREN-based INR models. Spatial coordinates constitute the model inputs, while the corresponding radio measurements constitute the prediction targets. Separate representations are maintained for RSRP and SINR, allowing each model to learn the spatial distribution of the respective radio-quality indicator. Through this process, the sampled radio information generated by the Digital Twin is encoded into the parameters of the trained models.

Once trained, the models are made available to the Digital Twin Prediction Module for inference within the CoCo lifecycle. For each incoming session, the CAIO obtains the relevant user and session information, including the user's spatial location. These coordinates are provided to the trained RSRP and SINR models, which return the corresponding predicted radio-quality values. In this way, the CAIO can obtain an estimate of the radio conditions at the specific user location without requiring the complete Digital Twin dataset to be queried or reconstructed for every request.

The predicted values are evaluated against configurable signal-quality thresholds. In the current configuration, an RSRP above -110 dBm and an SINR above 0 dB are considered the minimum acceptable conditions. Both criteria must be satisfied for the connectivity assessment to be considered

successful. If this condition is met, the Digital Twin Prediction Module provides a positive indication to the CAIO and the CoCo lifecycle proceeds to the subsequent reasoning, scheduling, and Trust Function coordination stages. Otherwise, the radio conditions are considered insufficient and the request does not progress through the remaining trustworthiness provision workflow.

The figures below illustrate representative outputs of the Digital Twin Prediction Module for the NCSR “Demokritos” testbed, showing the spatial distribution of the predicted RSRP and SINR values, respectively. Each representation associates locations within the considered deployment area with the corresponding predicted radio-quality values, providing a continuous view of the expected signal conditions. The yellow dot indicates the antenna location, providing a spatial reference for interpreting the resulting radio maps.

Through this implementation, the sampled radio information generated by the Exata Digital Twin is transformed into a continuous predictive representation that can be directly queried by CoCo. The Digital Twin provides the underlying radio-environment data, the SIREN-based INRs learn the spatial distributions of RSRP and SINR, and the trained models expose these learned representations to the CAIO during operation. This maintains a clear separation between Digital Twin data generation, model training, and CoCo inference, while enabling connectivity-aware orchestration at the beginning of each user session.

The exact model architecture, training procedure, dataset generation process, and performance evaluation are further analyzed in the corresponding WP5 deliverables.



Figure 16: Spatial distribution of predicted RSRP values across the deployment area. The yellow dot indicates the antenna location

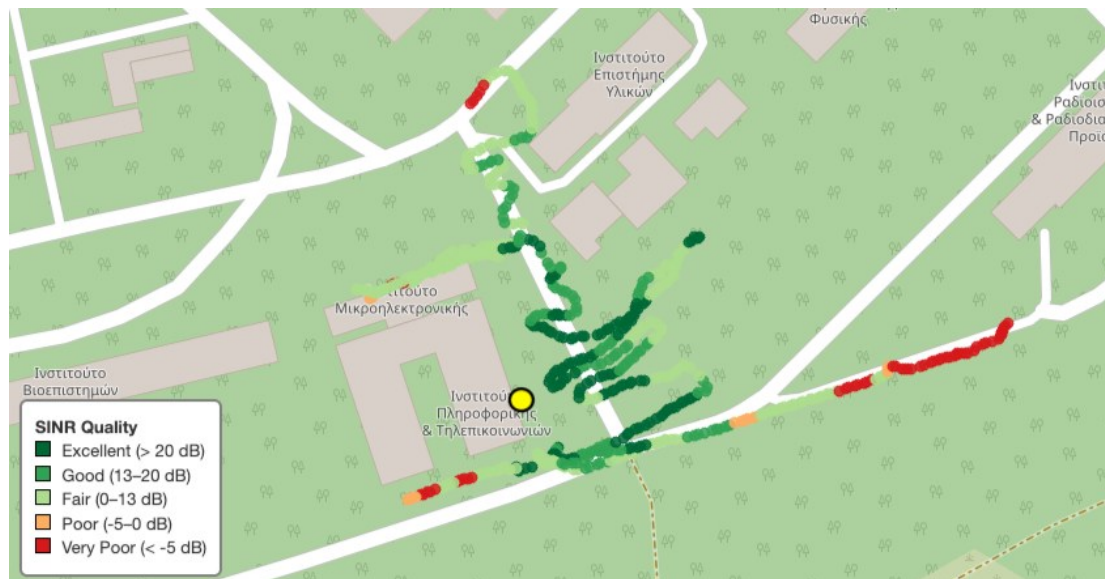


Figure 17: Spatial distribution of predicted SINR values across the deployment area. The yellow dot indicates the antenna location

4.3 MACHINE LEARNING MODULE

Once the received intents have been validated, the relevant user information has been stored or updated in the User Registry, and the CAIO has provided the indication to proceed after consolidating the Digital Twin Prediction Module, the Machine Learning Module is triggered to take over and derive the trustworthiness scores based on the user's needs. The structured intents received from the Chatbot are processed using a ***Bidirectional Encoder Representations from Transformers (BERT)*** [5] model, whose contextual representations are used to capture the meaning of the natural-language trustworthiness requirements. The model combines this representation with a routing process that associates each intent with the appropriate Trust Function and directs its BERT representation to the corresponding regression head in order to derive the scores per TF. Subsequently those scores are provided the rest of the inference steps in order to compute the nLoTw scores.

4.3.1 TRAINING

The model is implemented using ***PyTorch*** [6] and ***Hugging Face Transformers*** [7] and trained on the domain-specific trustworthiness dataset introduced in D4.1, publicly available in ***Zenodo*** [8]. Each dataset entry associates a natural-language trustworthiness expression with a corresponding Trust Function label and a trustworthiness score with a snippet of it presented in table 6. The initial entries were defined and evaluated by domain experts to represent trustworthiness requirements originating from vertical stakeholders, operators, system integrators and end users. A higher score indicates that an intent (keyword) indicates a stronger need for its respective trustworthiness dimension (class). For example, in the case of Privacy, the intent "*end-to-end encryption*" indicates a stronger demand for privacy over the system than "*social media information*". Additionally, a variety of intents populates the dataset, both simple network-agnostic intents that could originate from a simple everyday user and more specific demands originating from industry professionals. Taking cases such as this under consideration, enables CoCo's AI model and the broader SAFE-6G solution to provide value and importance to the user's interaction with the platform, regardless of his/her expertise.

Keywords	Score	Class
reliable application	1,40	Reliability
fast algorithms	1,40	Reliability
seamless operation	41,80	Reliability
social media information	1,60	Privacy
need encryption	61,80	Privacy
end-to-end encryption	81,10	Privacy
virus scanning	50,85	Security
data breach	90,10	Security
load balancing	1,33	Resilience
traffic prioritization	34,67	Resilience
critical resources	71,10	Safety

Table 6: Snippet of ML model's training dataset

The model architecture itself, uses **BERT** as a shared contextual representation (embeddings) layer, followed by five Trust Function-specific **regression heads**, one for Safety, Security, Privacy, Resilience and Reliability, as illustrated in Figure 18. The Trust Function label (class) associated with each dataset expression is used to route BERT's contextual embeddings to the corresponding Trust Function-specific regression head during training. In this way, the common BERT representation captures the semantic information of the expressions, while the individual regression heads learn the dimension-specific score mappings.

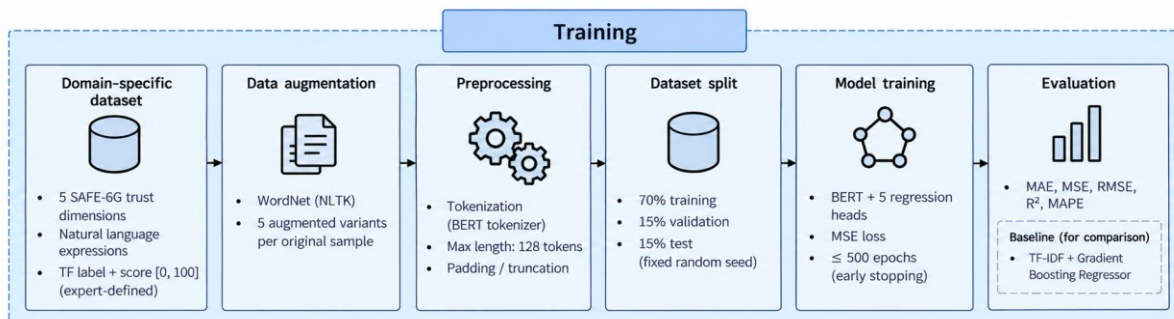


Figure 18: ML model's training process

During preprocessing, the dataset's trust expressions(keywords) are tokenized using the **BERT tokenizer** with a maximum sequence length of 128 tokens, applying padding and truncation where required. The trustworthiness scores are *normalized* before training, allowing the regression task to be learned on a continuous numerical scale. The training dataset is additionally augmented using **NLTK WordNet-based** [9] synonym replacement, generating five augmented variants for each original sample. This increases the linguistic variation of the training data and reduces the model's dependence on the particular wording of the original trust expressions.

Diving into the specifics of the training, the process uses the **AdamW optimizer** [10], which is well suited for fine-tuning transformer-based models such as BERT. AdamW decouples weight decay from the gradient update, allowing regularization to be applied more effectively than in standard Adam. In the current implementation, weight decay is applied during optimization to penalize overly large

parameter values and reduce the risk of overfitting. Additionally, **gradient clipping** is used during backpropagation by limiting the maximum gradient norm to a predefined value. This helps prevent unstable updates caused by excessively large gradients and contributes to a more stable training process, especially when fine-tuning the shared BERT encoder together with the Trust Function-specific regression heads.

The dataset is divided into 70% training, 15% validation and 15% test data, using a fixed random seed for reproducibility. The model is trained using **Mean Squared Error (MSE) loss** for a maximum of 500 epochs, with early stopping based on the validation loss. More specifically, training stops when validation loss does not improve a predefined number of epochs, assisting in overfitting prevention. The model achieving the best validation performance is retained for subsequent testing and inference. As presented in D4.1, its performance is evaluated using **MAE, MSE, RMSE and R^2** . A baseline combining *TF-IDF* features with a *Gradient Boosting Regressor* is also considered to provide a reference for evaluating the contribution of BERT’s contextual representations. The metrics produced during the evaluation are presented in the table below:

Metric	Baseline Model	BERT
Mean Absolute Error (MAE)	0.138	0.055
Mean Squared Error (MSE)	0.027	0.006
Root Mean Squared Error (RMSE)	0.164	0.076
R^2 score	0.548	0.907

Table 7: Metrics produced during the evaluation of ML model and its Baseline

In parallel, the following plots provide a more detailed view of the learning behaviour of the model throughout the training process. Figure 19 presents the evolution of the training and validation losses, expressed as Mean Squared Error (MSE). Both losses decrease rapidly during the first epochs, indicating that the model learns the underlying relationship between the input representations and the target trustworthiness scores at an early stage of training. After approximately five epochs, the rate of improvement becomes more gradual and both curves converge towards low and relatively stable values. Importantly, the validation loss follows the same overall trajectory as the training loss, without a growing separation between the two curves. This behaviour indicates that improvements on the training data are also reflected on unseen validation samples and provides no clear indication of overfitting within the illustrated training period.

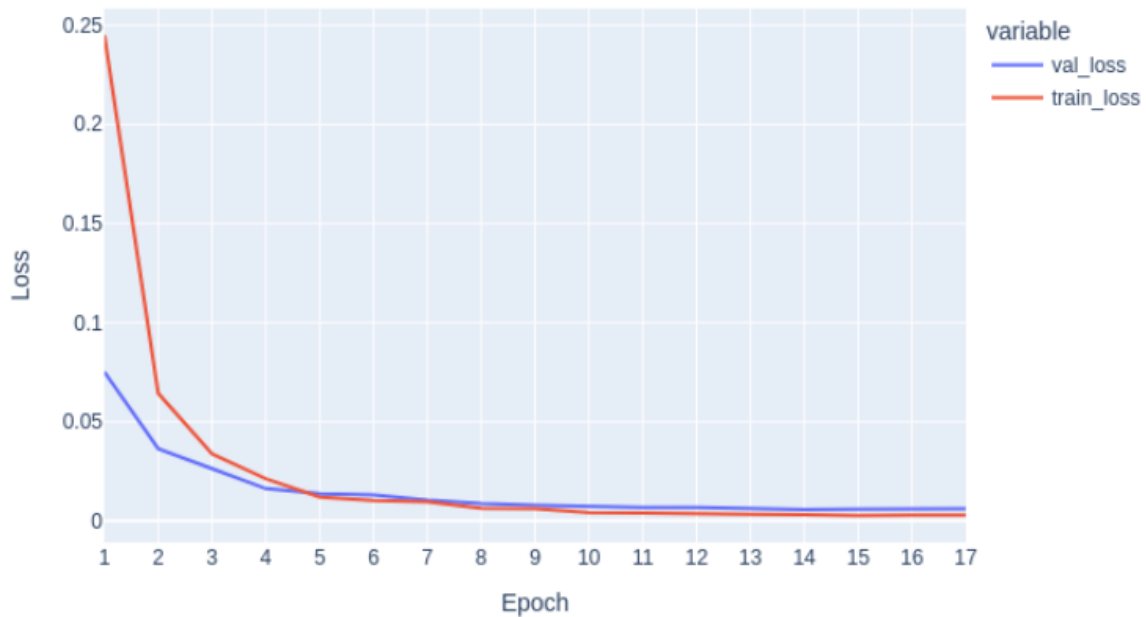


Figure 19: Validation and Training Loss per epoch

Figure 20 complements the loss analysis by presenting the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) calculated on the validation set. Both metrics decrease substantially during the initial epochs and continue to improve more gradually as training progresses. The MAE decreases from approximately 0.22 to around 0.05, indicating a considerable reduction in the average magnitude of the prediction errors. Similarly, the RMSE decreases from approximately 0.28 to around 0.08. Since RMSE gives greater weight to larger prediction errors than MAE, the simultaneous reduction of both metrics suggests that training reduces both the typical prediction error and the occurrence or magnitude of larger deviations. The curves begin to stabilize during the later epochs, indicating that the model is approaching convergence and that additional training produces comparatively smaller improvements.

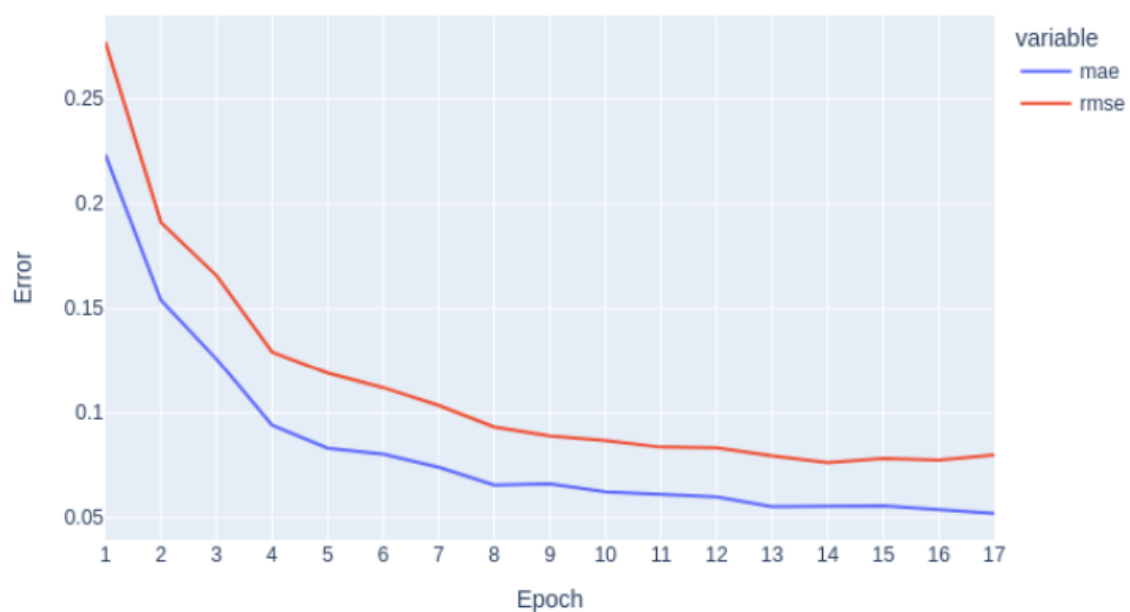


Figure 20: MAE and RMSE per epoch

Figure 21 shows the evolution of the coefficient of determination (R^2) on the validation set. The R^2 value improves sharply during the first epochs, moving from a negative initial value to above 0.7 by approximately the fourth epoch, before increasing more gradually towards values close to 0.9. The negative value observed during the first epoch indicates that the model initially provides a poor fit to the validation data. The rapid increase during subsequent epochs demonstrates that the learned representations progressively capture the relationship between the input intents and their target scores. The stabilization of R^2 around 0.9 in the later epochs indicates that the model explains a large proportion of the variance in the validation targets, while the small fluctuations towards the end are consistent with the plateau observed in the error and loss metrics.

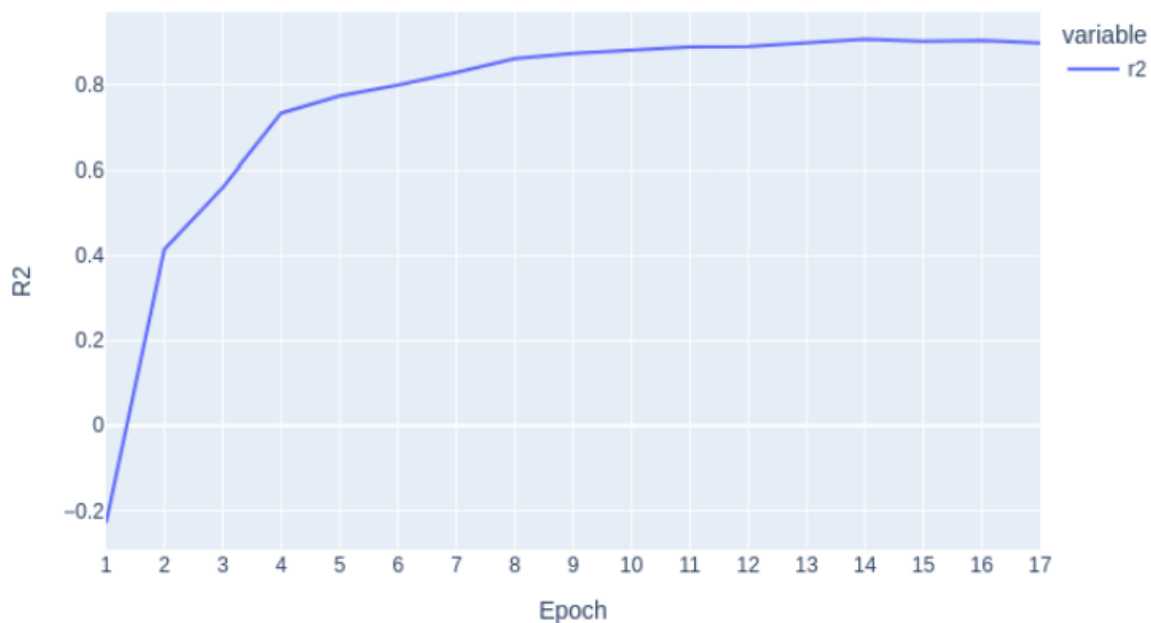


Figure 21: R squared per epoch

4.3.2 INFERENCE

During inference, the best-performing trained model is used to estimate the trustworthiness score associated with each received intent. Each intent is encoded by the shared BERT model to produce a contextual representation, and the Trust Function label associated with the intent routes that representation to the corresponding Trust Function-specific regression head. The selected head predicts a score in the range $[0, 100]$ for the individual intent. Consequently, different intents within the same user interaction may be processed by different regression heads, depending on the trustworthiness dimension they represent. In the implemented inference pipeline, the trained model weights are loaded once by the inference handler when the CoCo service starts. The handler initializes the same BERTForQuantification architecture used during training, loads the saved best_model.pth parameters, moves the model to the available execution device, and switches it to evaluation mode. Each received intent is tokenized with the bert-base-uncased tokenizer using the same maximum sequence length as during training. Inference is then performed without gradient calculation because model parameters are not updated during runtime. The individual regression-head outputs are subsequently processed by the intent-based weighting and normalization steps to derive the final nLoTw scores.

A user interaction may contain several intents belonging to the same or different trustworthiness dimensions. The number of intents associated with each dimension is therefore considered when consolidating the individual predictions. For example, three Security-related intents and one Reliability-related intent indicate a stronger presence of Security requirements within the overall interaction.

Following the weighting approach established in D4.1, CoCo derives intent-based weights from the number of intents associated with each trustworthiness dimension. These weights are applied when consolidating the individual regression outputs, producing a common five-dimensional representation of the user's trustworthiness requirements. The resulting normalized Level of Trustworthiness (nLoTw) contains the values corresponding to [Safety, Security, Privacy, Resilience, Reliability]. For example, an nLoTw of [0.0, 32.7, 0.0, 39.99, 0.0] represents the inferred trustworthiness requirements across the five dimensions for a particular user interaction. The complete inference process is illustrated in Figure 22.

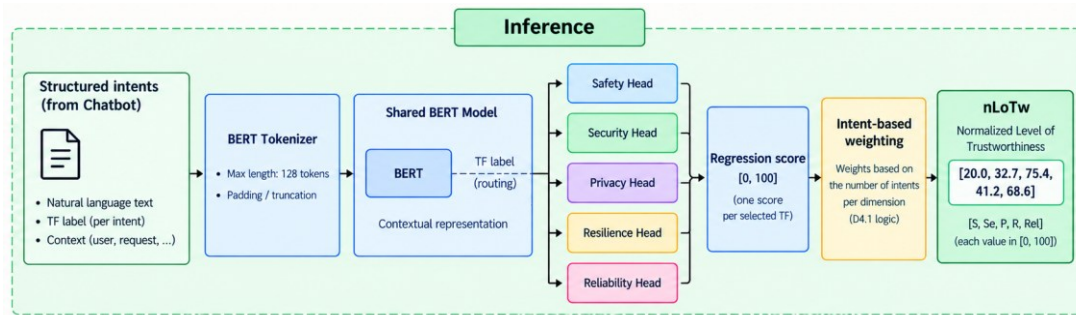


Figure 22: ML model's Inference process

Diving deeper into the calculation of the final scores, it is important to explain the process occurring right after the regression is completed and the formulas used. The computation of nLoTw is mathematically bound to the computation of the weights.

The number of sentences n_{TFj} classified under each trust function TFj , for $j \in \{1, 2, 3, 4, 5\}$ and the total number of sentences $\mathbf{N}$ is used to define a dynamic weighting mechanism that reflects the distribution of user concerns. Instead of using simple frequency ratios, the SAFE-6G model introduces a logarithmic-based weighting function that improves smoothness and mitigates sparsity effects:

$$W_{TFj} = \frac{\log(1 + n_{TFj})}{\sum_{j=1}^5 \log(1 + n_{TFj})} \quad (1)$$

This weight captures the proportional relevance of each TF in the user's intent profile while $\log$ transform is used to curb dominance by very large counts. In essence, it applies diminishing returns, letting repeated mentions of a TF initially contribute a lot to the weight calculation, while as n_{TFj} grows, the contribution starts to fade away. Practically, logarithmic weighting says, "more mentions means more importance, but repeated mentions should not linearly dominate the trust calculation."

The nLoTw is then computed as the normalized weighted average of the TF-specific regression outputs:

$$nLoTw = \frac{\sum_{j=1}^5 W_{TF_j} \cdot REG_{TF_j}}{\sum_{j=1}^5 W_{TF_j}} \cdot 100 \quad (2)$$

where REG_{TF_j} is the regression score per TF predicted by the model's inference. The resulting score serves as the system's initial estimation of the *overall* trustworthiness requirement by the user. Moreover, the overall score is decomposed into per-function contributions, the nLoTw scores per TF, the ones that are subsequently published to the TFs via the Message Broker:

$$nLoTw_j = \frac{W_{TF_j} \cdot REG_{TF_j}}{\sum_{j=1}^5 W_{TF_j}} \cdot 100 \quad (3)$$

So that:

$$nLoTw = \sum_{j=1}^5 nLoTw_j \quad (4)$$

This mechanism offers a dynamic way to reflect the structure of the user request in the final trustworthiness estimation. Trust Functions that are mentioned more frequently in the classified intents receive a higher contribution, since they represent stronger or repeated concerns in the interaction. At the same time, logarithmic weighting prevents dimensions with many intents from dominating the result excessively. This is useful when a user provides multiple similar intents for the same Trust Function, as it acknowledges their presence without suppressing the relevance of the other active dimensions.

Before the resulting nLoTw is forwarded to the subsequent components, the user-role-based cap is applied by the CAIO using the role maintained in the User Registry. The resulting scores are then mapped to the Trust Function deployment flavors: Low [0.01–39.99], Medium [40.0–79.99], and High [80.0–100.0], which are subsequently communicated to the Trust Functions through the Message Broker. On the TFs' side, these flavors are mapped to specific actions that need to be performed in order to achieve such a state (e.g. medium flavor for Reliability TF may be mapped to a specific change in the QoS profile, which in essence means that the TF will need to fire a specific API towards the 5G core). Of course, those actions are not directly performed, as the TF's will compare them against inspected telemetry and other available information by utilizing their AI Agents and they will report back to CoCo their own predictions and understanding of the system. The flavors that will be deployed are the ones that will arise from the final calculation of the cLoTw scores in later stages in the lifecycle.

4.4 MESSAGE BROKER

The Message Broker is implemented using **Apache Kafka** and provides the asynchronous communication mechanism between CoCo and the five Trust Functions. Communication is organized through dedicated Kafka topics for the individual TFs and follows a publish/subscribe model. On the CoCo side, the Message Broker comprises a **Kafka Producer** and **Kafka Consumer**. The Producer publishes information from CoCo to the corresponding TF topics, while the Consumer listens for messages published back by the TFs. Kafka was selected due to its support for asynchronous, scalable and reliable message exchange, while its topic-based communication allows CoCo and the TFs to remain decoupled and operate independently.

At implementation level, the Kafka Producer encodes outgoing messages as JSON payloads and publishes them to the appropriate TF-specific topics using the active session_id as the message key. This enables messages belonging to the same CoCo interaction to be correlated across the evaluation and deployment phases while preserving the ordering of records assigned to the same Kafka partition. For each involved Trust Function, the Producer constructs a dedicated payload containing session context, user information, the requested flavor, and the corresponding trust score before publishing it to the relevant topic. The Kafka Consumer handles the reverse direction of the exchange. It runs as a background worker within a dedicated consumer group and subscribes to the feedback and outcome topics published by the TFs. Upon reception, each message is deserialized, validated against the fields required for its topic type, and routed to the corresponding CAIO session handler. Invalid or incomplete messages are rejected and logged rather than being applied to the orchestration state.

As reflected in the overall architecture and lifecycle description, the Message Broker is used to publish the nLoTw scores, collect TF feedback, publish the conflict-free cLoTw scores, and collect the resulting TF deployment outcomes. In parallel, it informs the CAIO when these communication stages have been completed, thereby supporting session governance. Figure 23 illustrates this bidirectional topic exchange, while Table 8 summarizes the detailed topic names and payloads.

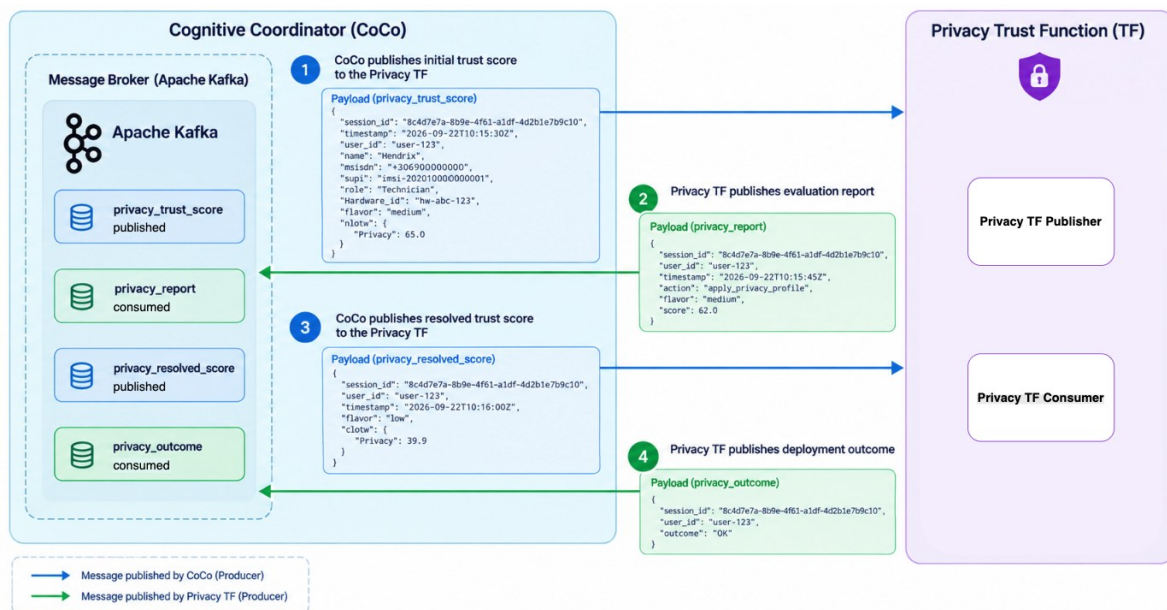


Figure 23: Bidirectional engagement of the Message Broker's communication with the Privacy TF

Producer	Consumer	Topic Name	Payload Fields
CoCo	TFs	<tf_name>_trust_score	session_id, timestamp, user_id, name, msisdn, supi, role, Hardware_id, flavor, nlotw
TFs	CoCo	<tf_name>_report	session_id, user_id, timestamp, action, flavor, score
CoCo	TFs	<tf_name>_resolved_score	session_id, user_id, timestamp, flavor, clotw
TFs	CoCo	<tf_name>_outcome	session_id, user_id, outcome

Table 8: Topic names and Payloads produced and consumed by the Message Broker

4.5 REASONING ENGINE

The Reasoning Engine is the rule-based processing component of CoCo and combines a Knowledge Base with the Conflict Resolution Manager (CRM) and Service and Function Scheduling capabilities. Its operation follows a facts-and-rules approach in which information collected during the lifecycle is evaluated against predefined cross-TF knowledge to derive a compatible final configuration. The reasoning process is triggered once the expected feedback from the involved TFs has been collected through the Message Broker. It processes the proposed scores and flavors, resolves incompatibilities, and produces the calibrated Level of Trustworthiness (cLoTw) together with the corresponding deployment configuration. The current implementation builds on the conflict-resolution logic presented in D4.1, replacing hard-coded rule evaluation with a custom Datalog-style engine. This allows the current configuration to be evaluated declaratively and enables conflicts and resolution decisions to be derived repeatedly as that configuration changes. The following subsections describe the two constituent parts of the Reasoning Engine and clarify the distinction between facts and rules.

- *Facts represent the current trustworthiness configuration kept in the session with support from the CAIO, including the nLoTw scores and the scores and flavors proposed by the TFs.*
- *Rules populate the Knowledge Base and describe the conditions under which the available facts may indicate a conflict or require a particular resolution action.*
- *The Conflict Resolution Manager utilizes its Datalog Engine to compare facts and rules and derive conflict-free configurations.*

4.5.1 KNOWLEDGE BASE

The Knowledge Base maintains the predefined cross-TF knowledge used during conflict resolution. It captures combinations of Trust Function flavors that cannot coexist within the same deployment, together with the constraints and resolution rules that must be considered when deriving the final trustworthiness configuration. For example, a High Security and High Reliability combination may introduce conflicting deployment requirements within the 5G Core, hence they cannot be deployed together. Examples of the conflicting combinations that populate the Knowledge Base as identified within SAFE-6G are implemented in plain **python** and are presented in Figure Figure 24.

```
CONFLICT_RULES: List[ConflictRule] = [
    # HIGH-HIGH
    ConflictRule("R001", frozenset(["Security", "Reliability"])), "high", "high"),
    ConflictRule("R002", frozenset(["Security", "Resilience"])), "high", "high"),
    ConflictRule("R003", frozenset(["Security", "Safety"])), "high", "high"),

    # MEDIUM-MEDIUM
    ConflictRule("R012", frozenset(["Reliability", "Resilience"])), "medium", "medium"),
    ConflictRule("R013", frozenset(["Reliability", "Privacy"])), "medium", "medium"),
    ConflictRule("R014", frozenset(["Resilience", "Privacy"])), "medium", "medium"),

    # CROSS LEVEL
    ConflictRule("R015", frozenset(["Privacy", "Security"])), "high", "medium"),
    ConflictRule("R016", frozenset(["Safety", "Reliability"])), "high", "medium"),
    ConflictRule("R017", frozenset(["Resilience", "Safety"])), "high", "medium"),
    ConflictRule("R018", frozenset(["Reliability", "Resilience"])), "high", "medium"),
```

Figure 24: Knowledge Base populated with Conflict Rules, implemented in python

The Knowledge Base assists the reasoning process but remains separate from the CRM's implementation. This separation allows the conflict knowledge to be maintained independently of the engine's evaluation mechanism. Additional conflicting combinations or cross-TF constraints can be incorporated into the Knowledge Base without requiring the overall reasoning process to be rewritten as a sequence of individual conditional statements.

4.5.2 CONFLICT RESOLUTION MANAGER AND SERVICE FUNCTION SCHEDULING

The Conflict Resolution Manager is implemented through a custom Datalog-style rule engine developed from scratch within CoCo, without relying on an external rule-engine library. Once the facts have been collected (initial nLoTw scores and proposed scores and flavors after consumption of TF feedback), the CRM combines these facts with the rules maintained in the Knowledge Base to identify incompatible flavor combinations and derive the corresponding resolution actions. The engine supports comparison operators such as less-than, greater-than and equality checks and controlled negation, allowing rules to evaluate relationships between trustworthiness scores and conditions that depend on the absence of a particular fact, rather than relying exclusively on exact flavor matching.

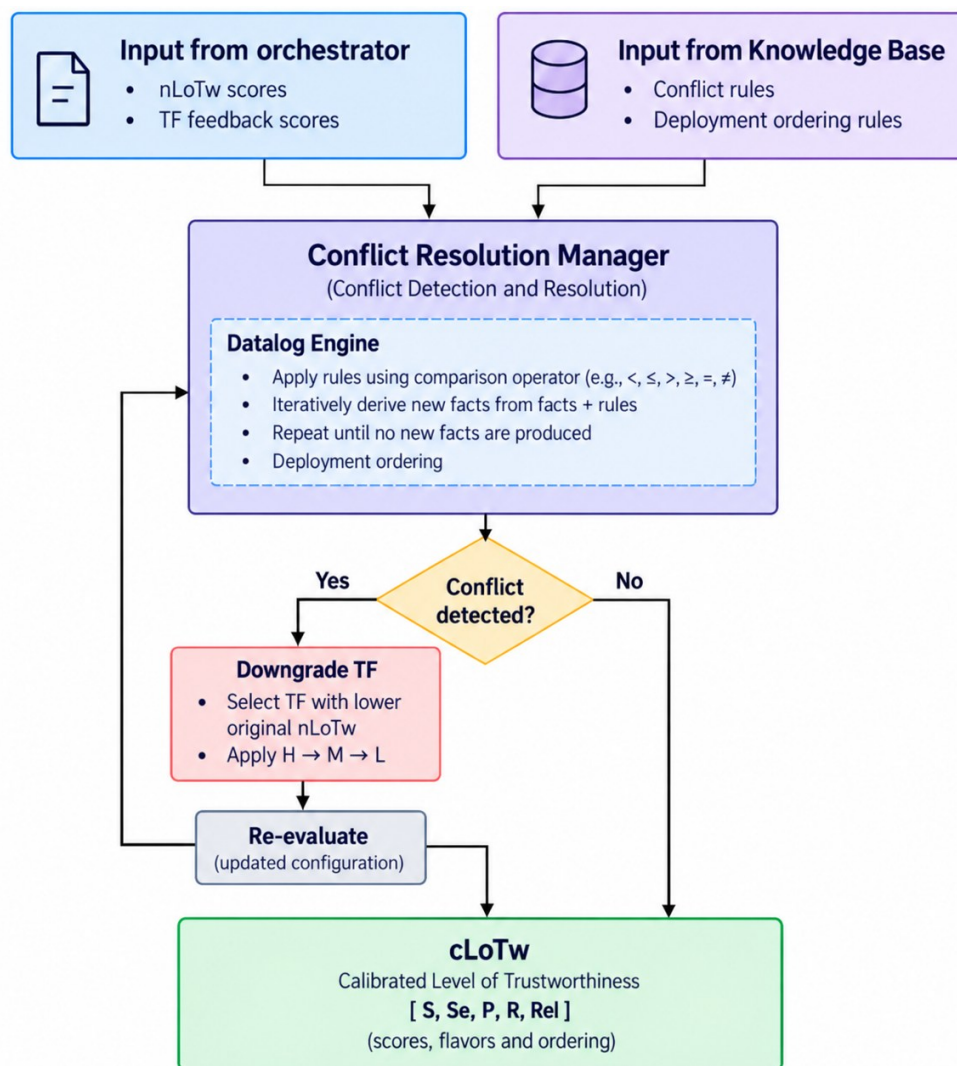


Figure 25: Reasoning Engine's workflow

The engine follows the bottom-up evaluation approach illustrated in Figure 25. Starting from the facts provided by the CAIO, it evaluates the applicable Knowledge Base rules and derives additional facts whenever their conditions are satisfied. Newly derived facts can participate in subsequent evaluations, and the process continues until a complete pass produces no additional facts. This provides a systematic mechanism for determining the conflicts and resolution actions implied by the current trustworthiness configuration. When a conflict is detected, the CRM uses the original nLoTw scores to determine which Trust Function should be adjusted. Following the logic established in D4.1, the TF with the lower initial nLoTw score yields and its proposed flavor is downgraded by one level. The updated configuration is then evaluated again, because resolving one conflict may expose another incompatibility involving a different TF. This iterative process continues until a stable, conflict-free configuration is obtained. Once the final flavors have been determined, the CRM adjusts the corresponding trustworthiness scores where necessary to respect their upper bounds. The resulting values form the cLoTw and, together with the final flavors, are used for the subsequent deployment stage. The Service and Function Scheduling capability then determines the order in which the involved Trust Functions should execute their deployment actions, taking into account the ordering rules and dependencies stored in the Knowledge Base. This allows the CAIO to coordinate the final configuration rather than issuing potentially conflicting deployment actions independently.

In summary, the CRM follows the conflict-resolution logic established in D4.1 while using the internal Knowledge Base through a Datalog-style engine that adds three important capabilities:

- a) **Declarative reasoning**, moving away from direct if/else processing and expressing conflict logic as facts and rules.
- b) **Bottom-up inference**. The engine derives additional facts from existing facts and rules and can then reuse those derived facts in subsequent reasoning. It repeatedly evaluates the rule set until no additional facts can be derived, providing richer behavior than a single predefined condition check.
- c) **Iterative conflict resolution**. After the CRM changes one TF configuration, the resulting state may interact differently with another TF. The rules are therefore evaluated again against the updated facts until the configuration stabilizes. This allows the engine to reason not only over exact flavor combinations but also over score relationships, missing facts, and explicit overrides.

4.6 EXPLAINABLE AI

4.6.1 OPERATIONAL CONTEXT

The operational context of the XAI Aggregator is summarized in Figure 26 and can be viewed as a cycle that begins and ends with the user. Once the user–Chatbot interaction is complete, the conversation is forwarded to the Cognitive Coordinator (CoCo), which orchestrates the trustworthiness-provision lifecycle. At the end of this lifecycle, after the trustworthiness scores have been computed and the deployment actions completed, CoCo automatically triggers the XAI module and provides it with the contextual information derived from the user interaction and the resulting Trust Function decisions.

In parallel, the XAI module draws on two sources of knowledge: the Trust Functions provide the domain description together with the model and feature descriptions, and the XAI methods catalogue provides the description of the available methods with their limitations and assumptions. The XAI module then produces the explanations and returns them to CoCo, which aggregates them together with the trustworthiness and outcomes into the final response delivered to the user through the Chatbot.

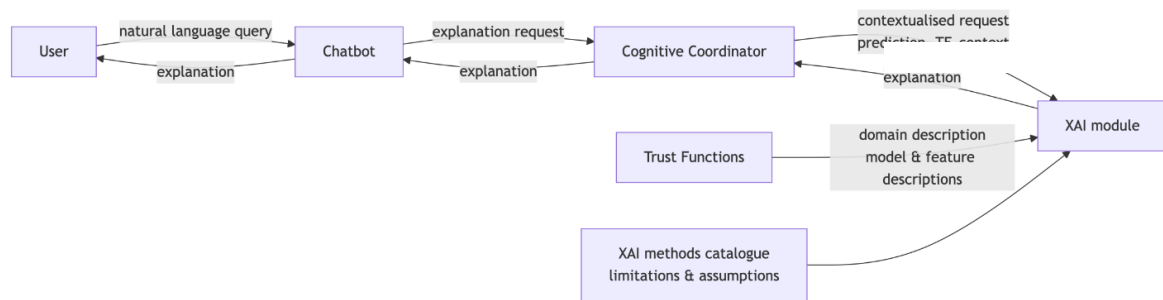


Figure 26: XAI - Overview of Operational Context

4.6.2 STRUCTURE OF THE XAI AGGREGATOR

The XAI Aggregator decomposes its activity into three main steps, each corresponding to a dedicated submodule, as summarized in Figure 27: the XAI Selector, XAI Executor, and XAI Interpreter. These submodules share internal resources describing the application domain and TF models, together with a catalogue of callable APIs corresponding to the available explainability methods. Individual XAI tools are hosted and invoked through the MLOps module, which provides on-demand computing resources and access to the AI models of the Trust Functions.

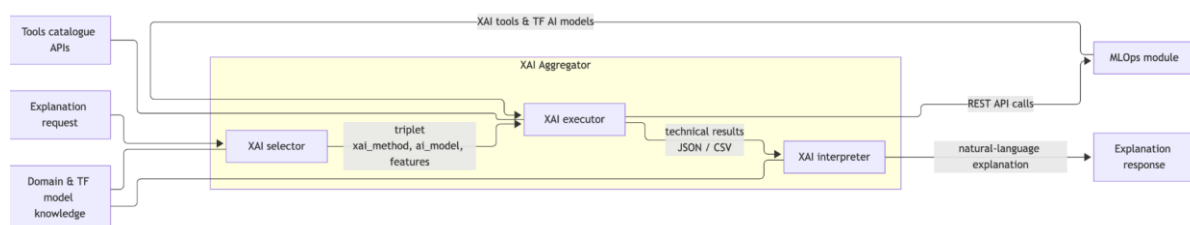


Figure 27: XAI - Decomposition of Sub-Modules

4.6.2.1 DOMAIN & TF MODEL KNOWLEDGE

The XAI Aggregator relies on a catalogue of explainability methods, each of which is described by a structured entry capturing its applicability domain and its methodological assumptions. Each catalogue entry carries more than a name: it specifies the explanation type (local, global or both), the supported task types (classification, regression), the input feature structure (tabular, text), whether the model must be differentiable, the compatible model architectures, and the methodological assumptions underlying the interpretation of the method. These assumptions are exploited by the XAI interpreter.

This knowledge base accompanies the explanation from its selection to its final formulation. The selector relies on the model description (task type, input feature structure, differentiability and the architecture) to filter the catalogue and retain only the methods applicable to the model at hand. The executor uses the feature descriptions to build the REST queries and to resolve the data resources referenced by the model descriptor, such as the training dataset, the background dataset or, for CEM and Prototype Counterfactuals, the pre-trained autoencoder. Finally, the interpreter combines the assumptions of the selected method with the natural-language description of each feature to produce an explanation that is both grounded in the actual reasoning of the model and understandable by the final user. The same piece of knowledge is thus reused at different stages for different purposes, which is why the sub-modules share it as a common internal resource, as illustrated in Figure 28.

4.6.2.2 XAI SELECTOR

The selector is responsible for choosing one explainability method to run on the Trust Function AI models. It combines the information about the AI models of the Trust Functions with the prediction context provided by CoCo to decide which technique is appropriate, and it outputs a triplet $\langle \text{xai_method}, \text{ai_model}, [\text{features_list}] \rangle$ describing the method to apply, the model to explain, and the features of interest. The selection logic is hybrid: it first determines whether a local or a global explanation is required, then filters the catalogue of available methods according to the characteristics of the model (task type, input data type, differentiability, architecture), and finally ranks the remaining candidates to pick the best fit.

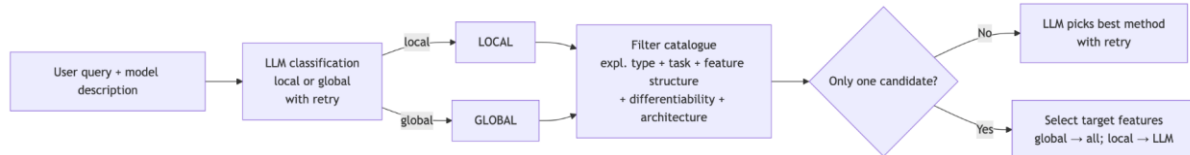


Figure 28: XAI Selector Sub-Module

Stage 1 — Explanation-type detection. The selector first classifies the request as requiring a local or a global explanation. The classification is performed by an LLM prompted with the context provided by CoCo and the model description; the prompt is re-issued up to three times with progressively simpler instructions when the answer is not usable. If the LLM still cannot produce a valid classification, an explicit selection error is raised, there is no keyword-based matching and no heuristic fallback.

Stage 2 — Applicability filtering. The selector then filters the catalogue according to five characteristics of the model: the explanation type (local or global), the task type (classification or regression), the input feature structure (tabular or text), the model differentiability, and the compatibility of the model architecture (tree-based keywords enable Tree SHAP, neural-network keywords enable Integrated Gradients, CEM and Prototype Counterfactuals).

Stage 3 — Selection. If a single method remains, it is returned directly. Otherwise, an LLM selects the best method among the remaining candidates, prompted with the explanation type and the description of each candidate method. The prompt is re-issued up to three times when the answer is

not usable. If the LLM returns an unrecognized method name, an explicit selection error is raised, there is no question-pattern ranking, no keyword-based override and no fallback to a pre-ranked candidate.

Stage 4 — Target-feature selection. Finally, the selector determines which features are of interest. As in Stage 1, this determination does not rely on a user query: for a global explanation, all features of the model are retained; for a local explanation, the features of interest are identified from the model descriptor and the context provided by CoCo, with the assistance of the LLM when required. The case where a user asks for a description of the features themselves no longer applies in the automatic flow, since the user does not address explanation-related questions to the XAI module.

4.6.2.3 XAI EXECUTOR

Once an XAI method has been selected, the XAI Executor triggers the corresponding computation through the MLOps module. The XAI tools available in the catalogue are exposed as services through a REST API. The executor constructs the required REST requests using the API documentation and domain description, triggers the selected explainability computation, waits for the relevant parallel executions to complete, and collects their results. The outputs are returned in structured formats such as JSON or CSV, while their precise content depends on the selected tool.

Two catalogue entries require a representative background dataset: Tree SHAP and Kernel SHAP. The background dataset provides a baseline of typical feature values against which the attributions are computed, and its representativeness directly conditions the reliability of the resulting explanations. Because both methods build on the same notion of a reference distribution, the choice between them is essentially a trade-off between performance and generality: Tree SHAP when the model is tree-based and speed matters, Kernel SHAP when the model cannot be handled by a specialised explainer. In both cases, a poorly chosen background dataset can bias the attributions, which is why the interpreter flags the background dataset size in its assumption-verification step.

4.6.2.4 XAI INTERPRETER

The XAI Interpreter first interprets the technical results produced by the selected XAI methods. It relies on the method descriptions in the XAI catalogue and on the documented API return types to generate a technically grounded explanation. Because explanations based directly on model features may be difficult for an end user to interpret, the module subsequently uses an LLM to reformulate the technical output into a more accessible explanation. This interpretation is grounded in the descriptions of the application domain and of the observed and predicted features.

Because XAI methods rely on methodological assumptions that, when violated, can make an explanation misleading, the interpreter incorporates an automatic assumption verification step. Each method in the catalogue is associated with the assumptions underlying its interpretation, for example, Permutation Importance assumes feature independence, and Kernel SHAP assumes a representative background dataset. These assumptions are injected into the LLM prompt as an "assumptions to verify" section, and the interpreter instructs the model to emit a CAUTION line whenever an assumption is at risk of being violated.

In addition, concrete data signals are computed automatically from the available data, such as feature correlation, background dataset size and anchor coverage, and provided to the LLM so that its verification is based on actual metrics rather than on generic instructions. The CAUTION lines detected in the LLM response are extracted and surfaced to the user together with the explanation, so that the user is informed whenever the explanation may not be fully reliable.

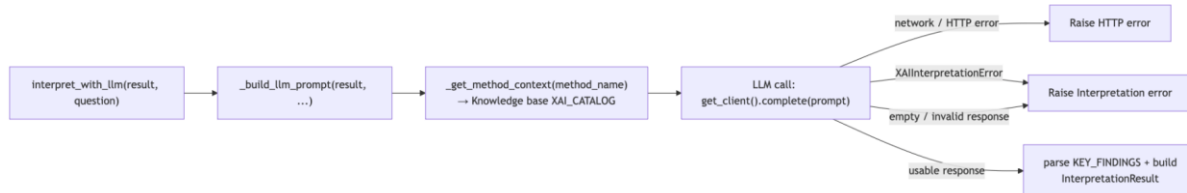


Figure 29: XAI Interpreter Sub-Module

To produce accurate explanations, the interpreter relies on a rich description of the environment in which the model operates. The prompt given to the LLM includes the user's question, the description of the model and of its task, the description of the output feature and of the actual prediction, and the per-feature attributions produced by the selected method together with the natural-language description of each feature. By providing this contextual information, the interpreter improves the LLM's understanding of the system, which supports the factual accuracy of the generated explanations and allows the interpreter to answer a wide range of question types, whether they concern a specific prediction, the influence of a particular feature, or the general behaviour of the model.

4.6.3 EVALUATION MODULE

The selector, which maps available model information to an XAI method, and the interpreter, which turns technical attributions into plain-language explanations, are validated through a dedicated CLI evaluation harness. It runs the real selection and interpretation pipelines on gold cases, reporting accuracy and quality metrics together with failing cases for diagnosis.

The harness remains external to the core pipeline and does not alter the runtime behavior of the selector, executor, or interpreter. Each run records its reproducibility context (random seed, LLM identifier, decoding parameters, and environment versions) so results remain traceable and comparable across models and evaluation-set revisions.

4.6.4 XAI AGGREGATOR DEPLOYMENT

4.6.4.1 INTRODUCTION

The XAI Aggregator is deployed as a self-contained service hosting the complete XAI pipeline, composed of the selector, executor and interpreter. It exposes a FastAPI REST interface through which explanations can be requested using either a natural-language question (NLP mode) or an explicitly selected XAI method (direct mode). The service provides a POST /explain endpoint for explanation requests and a GET /health endpoint for health monitoring. Interactive API documentation is also automatically exposed by FastAPI.

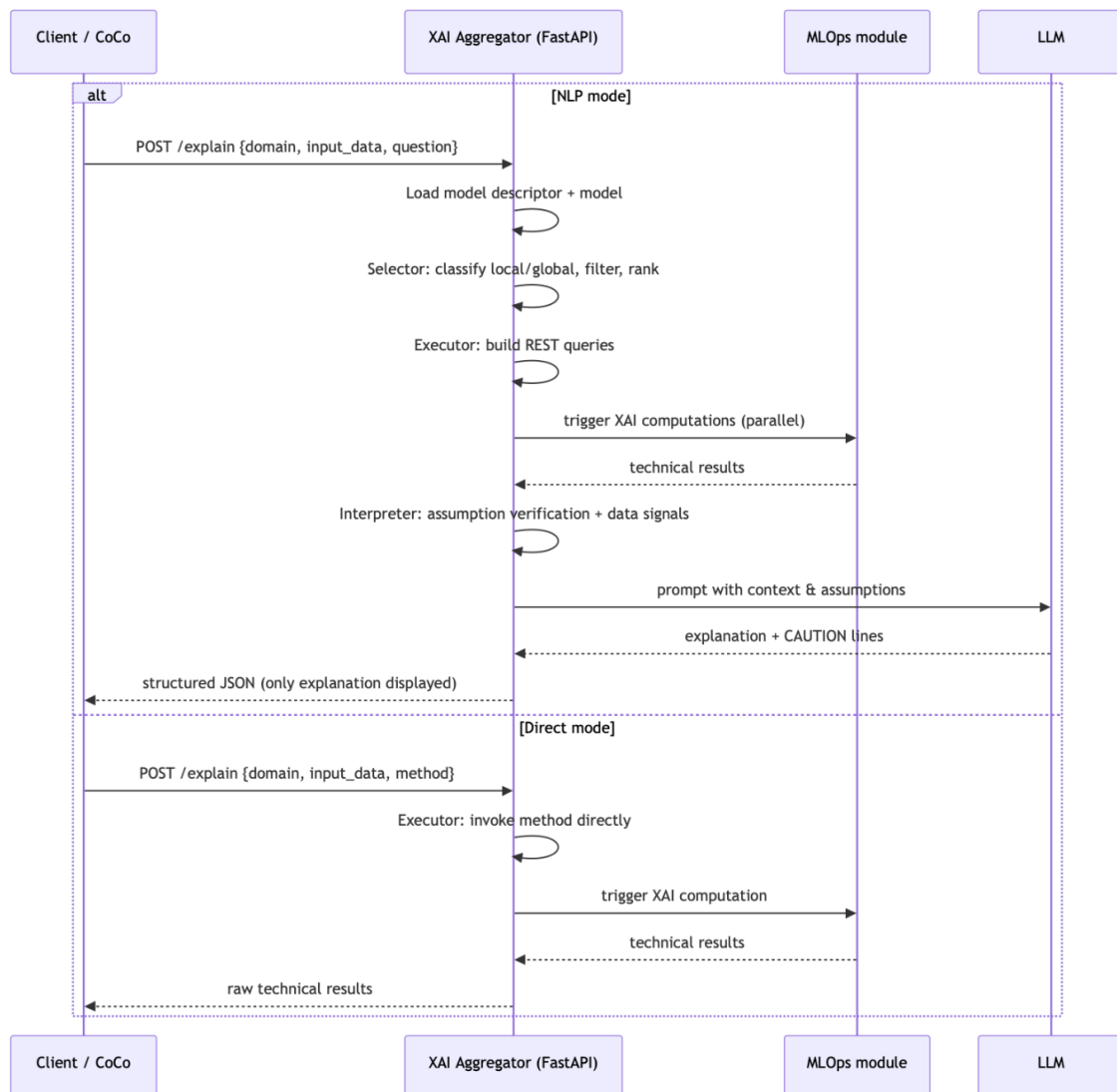


Figure 30: XAI Aggregator sequence diagram

API is intentionally kept independent from the complete description of the AI model being explained. A request identifies the target Trust Function model through a domain identifier. The corresponding model descriptor is then resolved server-side from the model registry. This descriptor contains the information required by the selector, executor and interpreter, including the model architecture, task, feature descriptions and location of the model artefact.

In production deployments, model artefacts are stored in MinIO object storage. The model registry therefore acts as the metadata layer associating a domain with its descriptor and the corresponding model stored in MinIO. When an explanation is requested, the XAI Aggregator resolves the domain, retrieves the associated descriptor and accesses the required model artefact from MinIO. This separates the public API from model storage and avoids transferring model descriptors or model files as part of individual explanation requests.

The POST /explain endpoint accepts the request payload presented in Table 9 and supports the two operating modes described below.

Field	Type	Description
domain	string	Operational domain identifying a model registered in the model registry.
question	string null	Natural-language question to be interpreted by the XAI pipeline in NLP mode. Optional when method is provided.
input_data	object array	Input feature values of the instance to explain. Keys must match the feature names defined in the model descriptor. A list can be provided to explain several instances in one request.
method	string null	XAI method to invoke explicitly in direct mode, e.g. shap, ale or permutation_importance.
explanation_type	string null	Optional override specifying a local or global explanation.
parameters	object null	Optional method-specific parameters.

Table 9: POST /explain request payload

4.6.4.2 NLP MODE

NLP mode is the default operating mode of the XAI Aggregator. The request contains a domain, the corresponding input_data and, optionally, a natural-language question. The complete XAI pipeline is then executed automatically.

The selector analyses the model descriptor retrieved from the registry together with the prediction context provided by the caller. It determines the explanation type, identifies the relevant features and selects an appropriate method from the XAI catalogue. The executor subsequently applies the selected XAI method to the target model, after which the interpreter converts the technical results into a natural-language explanation.

This mode corresponds to the interaction path in which the XAI Aggregator is automatically triggered by the Cognitive Coordinator (CoCo) at the end of the trustworthiness provision flow. The SAFE-6G Chatbot does not identify requests concerning the explanation of a Trust Function decision: the user does not ask for explanations. Once the trustworthiness and deployment stages have been completed, CoCo invokes the XAI Aggregator on behalf of the user, providing the information required to identify the corresponding domain and prediction context. The XAI Aggregator then processes the request and returns the resulting explanation to CoCo, which aggregates it into the final response delivered to the Chatbot.

The normal behaviour can optionally be constrained using explanation_type, for example when the caller explicitly requires a local or global explanation. The selection of the XAI method itself remains the responsibility of the selector in NLP mode.

The resulting response contains both the natural-language explanation and structured information describing the execution, including the method selected, the explanation type and the resulting feature importance.

4.6.4.3 DIRECT MODE

Direct mode provides programmatic access to individual methods of the XAI catalogue without requiring a natural-language question.

In this mode, the request contains a domain, the `input_data` and an explicit method. The XAI Aggregator resolves the corresponding model through the registry and invokes the requested XAI method directly. The NLP-based method-selection stage is therefore bypassed, and the results can be consumed directly by another SAFE-6G component or by an external programmatic client.

The method identifier is resolved against the XAI catalogue using tolerant name matching, allowing canonical method names as well as supported abbreviations and case variations. For example, methods such as SHAP, ALE or Permutation Importance can be explicitly requested.

Method-specific configuration can be supplied using the parameters, while `explanation_type` can be used when the execution must explicitly target a local or global explanation. Direct mode is particularly useful for deterministic integrations, automated evaluation and situations in which the caller already knows which XAI method must be executed.

4.6.4.4 DEPLOYMENT

The XAI Aggregator is deployed as a containerised, self-contained service exposing its functionality through a API REST interface. The service hosts the selector, executor and interpreter and processes explanation requests synchronously. The deployment separates three main categories of resources:

- The **XAI Aggregator service**, which implements the selection, execution and interpretation pipeline;
- The **model registry**, which stores the metadata and descriptors associated with each operational domain;
- **MinIO object storage**, which stores the AI model artefacts used in production.

Consequently, model binaries are not transferred through the XAI API. A request only specifies the target domain. The Aggregator uses this identifier to resolve the corresponding model descriptor and model location through the registry and subsequently accesses the model artefact stored in MinIO. Local or statically accessible model files may still be used for development and testing deployments, while MinIO provides the production storage mechanism. This architecture keeps the API payload independent from the underlying storage mechanism and makes it possible to update model artefacts and their associated metadata without modifying clients consuming the explanation API. The model descriptor registered for each domain follows the structure presented in Table 10.

Attribute name	Data type	Description
<code>model_description</code>	string	Natural-language description contextualising the purpose and usage of the Trust Function model. Used by the interpreter when producing user-oriented explanations.
<code>model_location</code>	URI / resource reference	Reference to the model artefact. In production, this identifies the corresponding model stored in MinIO.

model_architecture	string	Description of the model architecture, such as neural network, tree ensemble or SVM.
model_differentiability	bool	Indicates whether the model is differentiable, which is required by some XAI techniques.
model_task	Enum[classification, regression]	Machine-learning task performed by the model. Used by the selector when identifying compatible XAI methods.
input_features	feature description[]	Description of the model input features. Used by the executor and interpreter.
output_feature	feature description	Description of the model output.
training_data / background_data	resource reference	Dataset references used by XAI methods requiring training or background samples, such as Anchors, Similarity or Kernel SHAP.

Table 10: Model descriptor schema registered per domain

Attribute name	Data type	Description
name	string	Name of the feature. It is used as the corresponding key in model prediction calls.
type	type	Data type of the feature, such as integer, float, string or enumeration.
description	string	Natural-language description providing semantic context for the feature and supporting the interpretation of XAI results.

Table 11: Feature description schema

The POST /explain endpoint returns a structured JSON response following the schema presented in Table 12. When input_data contains several instances, a corresponding list of responses is returned in the same order.

Field	Type	Description
success	bool	Indicates if the explanation request completed successfully.
domain	string	Domain associated with the explained model.
question	string null	Original natural-language question; null when no question was supplied.
prediction_value	any	Prediction generated by the model for the supplied input.
prediction_confidence	number null	Prediction confidence or probability when provided by the model.
method_used	string null	XAI method effectively executed.
explanation_type	string null	Type of explanation generated, typically local or global.
feature_importances	array	Structured feature-attribution results, including feature, importance, direction and description when available.
explanation	string null	Natural-language interpretation generated by the interpreter when applicable.
cautions	array	Methodological cautions emitted when assumptions associated with the selected XAI method may be at risk.
execution_time_ms	number	Total processing time of the explanation request in milliseconds.

confidence	number null	Confidence associated with the interpretation when available.
error	string null	Error information when success is false.

Table 12: POST /explain response

4.7 COCO AND XAI AS A WHOLE – INTERACTIONS AND DATA FLOW

At a high level, the XAI Aggregator produces explanations associated with the decisions taken by the framework for a given user request. It is triggered automatically by the Cognitive Coordinator (CoCo) after the trustworthiness and deployment stages of the user flow have been completed, and the resulting explanations are returned to the user through the Chatbot. The user does not need to submit a separate explanation request: explainability is generated automatically for each user flow and is delivered together with the trustworthiness results.

Integration rationale. CoCo is intentionally the only component that requests explainability for all Trust Functions involved in a user flow. This simplifies integration by allowing CoCo to aggregate the individual explanations before returning them to the Chatbot. The Chatbot forwards the user intent to CoCo, which enriches the context, triggers the XAI Aggregator on behalf of the user, and combines the resulting explanations with the trustworthiness and execution outcomes before delivering the consolidated response.

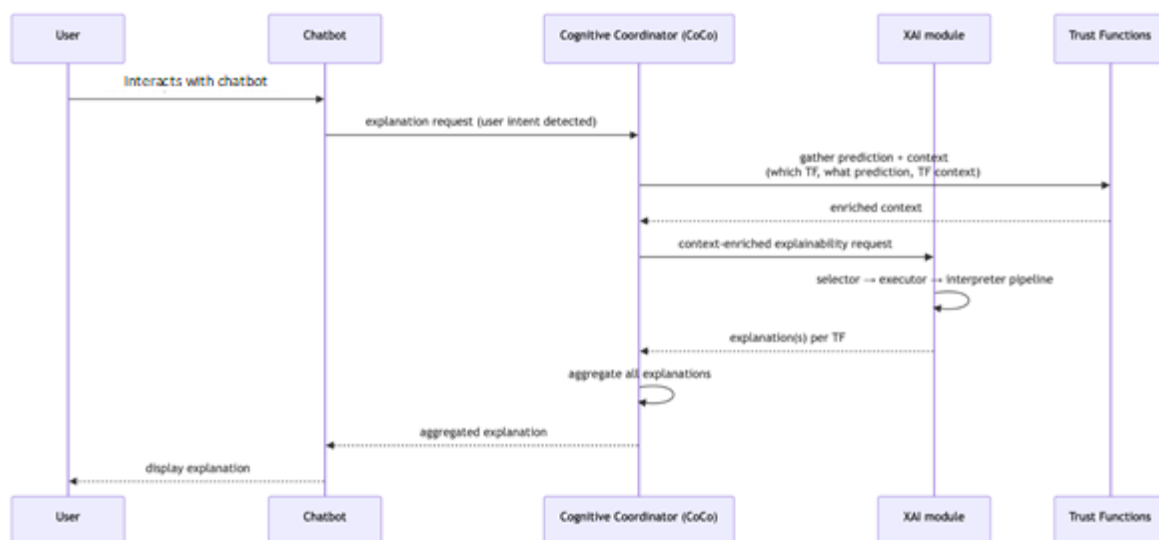


Figure 31: XAI - Orchestration Logic Sequence Diagram

4.8 XAI REQUIREMENTS COVERAGE

The design specified in this document is bound by the functional and non-functional requirements defined in D2.1. For the XAI component, five requirements have been defined. This section describes how each of these requirements is addressed by the design and implementation of the XAI Aggregator presented in this document.

4.8.1 COLLECTING INPUTS FOR THE XAI COMPONENT

The inputs of the XAI component are the outputs of the AI models associated with the Trust Functions. The result to be explained may be either a particular value to be analysed, such as an anomaly returned by the model, or the global behaviour of the model, for example how the model distinguishes low from high values. In both cases, the inputs concern each Trust Function individually; correlations between Trust Functions are deliberately not considered.

In the implemented XAI Aggregator, these inputs are collected through the POST /explain API exposed by the service. A request identifies, through a domain identifier, the Trust Function model whose decision is to be explained. The domain is resolved server-side against the model registry, which acts as an index of the registered Trust Function models and the locations of their artefacts.

For a local explanation, the request carries the input feature values of the instance to be explained. For a global explanation, the representative dataset used to analyse the model behaviour is resolved from the registry, using training or background data. In both cases, the prediction to be explained is produced by the Trust Function model itself at request time, ensuring that the module explains the actual decision of the corresponding Trust Function. Since each request targets a single domain, the inputs remain scoped to one Trust Function at a time, in line with the requirement.

4.8.2 BUILDING THE XAI MODELS

The XAI component provides two types of explanation for the Trust Function AI models: local explanations, which explain a particular value such as an anomaly or outlier returned by the model, and global explanations, which describe the overall behaviour of the model. To support these two levels of explanation, the XAI Aggregator implements a catalogue of twelve explainability methods. Each method supports local explanations, global explanations, or both. The catalogue, the individual methods, their applicability domains and their methodological assumptions are described in detail in the XAI Aggregator structure section and are therefore not repeated here.

The global methods are Accumulated Local Effects (ALE), Partial Dependence (PDP), Partial Dependence Variance (PDV) and Permutation Importance. The local methods are Anchors, the Contrastive Explanation Method (CEM), Counterfactual Explanations, Prototype Counterfactuals, Integrated Gradients and Similarity Explanations. Kernel SHAP and Tree SHAP support both local and global explanations.

Counterfactual explanations based on reinforcement learning are deliberately excluded from the catalogue because they target sequential and dynamic environments, which are outside the scope of the SAFE-6G use cases. The existing counterfactual methods already cover the cases considered in the project.

These methods originate from the XAI tools catalogue defined in the SAFE-6G specification and are implemented within the XAI module. Each method is represented by a structured catalogue entry describing its applicability domain and methodological assumptions. This information is used by the selector to choose an appropriate method according to the user query and the characteristics of the Trust Function model, as described in the XAI selector section.

4.8.3 SENDING RESULTS TO THE CHATBOT

The results produced by the XAI Aggregator are integrated into the SAFE-6G Chatbot through CoCo so that explanations can be communicated to end users. The interaction follows the orchestration described in Section 4.7. The Chatbot forwards the user intent to CoCo, which enriches the required context and triggers the XAI Aggregator on behalf of the user. The XAI Aggregator processes the request and returns the resulting explanation to CoCo, which combines it with the trustworthiness and execution outcomes before delivering the final response to the Chatbot.

The XAI Aggregator is exposed through the POST /explain API. The API takes as input a natural-language question and a domain identifier. The domain identifies the Trust Function model to be explained. Models are pre-registered in the model registry, which stores a descriptor for each model, including its purpose, architecture, task, features and the location of its artefacts. The XAI Aggregator therefore resolves the domain server-side and does not require the caller to know or transfer the model itself.

The API returns a structured JSON response containing the natural-language explanation generated by the interpreter together with the underlying XAI information. This includes structured feature attributions, with the feature name, importance, direction and description, as well as the method used, the explanation type, the prediction value and confidence, and any methodological cautions raised when an assumption of the selected method may not hold. This information is then communicated to the end user through the Chatbot to help interpret the results produced by the five Trust Function AI models.

4.8.4 PROPOSING DIFFERENT VISUALIZATIONS OF THE XAI RESULTS

Explainable results may take the form of text, numbers, tables, graphs, scatter plots or other visualizations. The XAI Aggregator supports several of these representations. The natural-language explanation produced by the interpreter is returned as text. Feature attributions are returned as structured numerical data, including the feature name, importance, direction and description, which can be rendered as tables by the consuming interface.

The technical outputs of the individual XAI methods, such as ALE effect curves, PDP results or SHAP values, are also produced in machine-readable formats such as JSON or CSV and can therefore be rendered as graphs or scatter plots by a visualization layer.

In the current implementation, the LLM used by the interpreter is text-only and not multimodal. No work has been carried out on LLM multimodality, and the LLM itself does not generate or return images, graphs or diagrams. Instead, the demonstration interface can generate basic method-specific graphs from the technical results returned by the XAI methods, including bar charts, effect curves and change visualizations.

These graphs are generated directly from the results of the executed method and are therefore consistent with the underlying explanation. They remain basic visualizations, however, and the most appropriate representation for a given method or result is not always guaranteed. The choice of the most suitable visualization for each case therefore remains an open point.

4.8.5 ALLOWING REAL-TIME INTERACTIONS WITH THE END-USER

Real-time interaction with the end user is supported through the SAFE-6G Chatbot, allowing users to interact with the XAI results. In the implemented XAI Aggregator, responsiveness is addressed at several levels.

Several explainability methods rely on background data, meaning labelled data representative of the training distribution of the Trust Function model. For these methods, approximately one hundred representative samples are considered a practical minimum for obtaining coherent results, while eight hundred to one thousand samples are the recommended order of magnitude. Providing more data than necessary generally brings limited benefit and can significantly increase execution time. This is particularly relevant for methods such as Kernel SHAP, whose computational cost increases with the number of background samples evaluated.

For this reason, the executor limits the amount of background and training data passed to these methods to a bounded number of representative rows. This keeps computation times compatible with interactive use while preserving a representative reference distribution.

Each explainability method is also subject to a hard execution timeout so that a slow or blocked computation cannot indefinitely delay the interaction. The total processing time of each explanation request is measured and returned in the API response, making request latency observable.

The module is additionally exercised through an evaluation harness that runs the LLM-based decision layers, namely the selector and the interpreter, on a set of gold cases and reports the resulting accuracy and quality metrics. The same harness can be used to compare different LLMs and assess the trade-off between response time and explanation quality, supporting the selection of a suitable model for interactive use.

The current implementation processes requests synchronously and does not yet manage concurrent requests. Interactions are therefore handled one at a time. This is sufficient for the current demonstration context but remains a known limitation to be addressed in a subsequent iteration.

5 FUTURE EVOLUTION OF CoCo AND ALIGNMENT WITH 3GPP INTENT-DRIVEN MANAGEMENT

The implementation presented in this deliverable has been developed primarily in the context of the SAFE-6G trustworthiness framework. In this configuration, CoCo receives structured trustworthiness requirements originating from the SAFE-6G Chatbot, interprets them through its AI capabilities, coordinates their assessment with the Trust Functions, resolves potentially incompatible configurations, supervises the corresponding deployment actions, and reports the resulting outcome to the user-facing component. Figures 2 and 9 demonstrate that these functions are already realized as an end-to-end cognitive coordination lifecycle rather than as isolated processing steps.

Although the current realization is trustworthiness-specific, the architectural principles adopted for CoCo are not inherently restricted to this domain. The separation of intent acquisition, cognitive interpretation, contextual assessment, conflict-aware reasoning, asynchronous coordination, enforcement, and reporting provides a basis for evolving CoCo into a generic cognitive coordination framework for intent-driven network management. Under this interpretation, the trustworthiness functionality developed in SAFE-6G represents the first operational profile of a broader CoCo architecture.

5.1.1 FROM THE SAFE-6G IMPLEMENTATION TO A GENERIC COGNITIVE COORDINATOR

The existing SAFE-6G implementation already exhibits the main abstractions required for such a generalization. The Chatbot and the CoCo API provide the intent-ingestion path; the Machine Learning Module performs cognitive interpretation; the User Registry and Digital Twin provide contextual information; the Trust Functions contribute domain-specific feasibility information and enforcement capabilities; the Knowledge Base and Conflict Resolution Manager provide conflict-aware reasoning; the Message Broker supports asynchronous interaction; and the CAIO governs the complete lifecycle and consolidates the final response. The genericization of CoCo therefore does not require replacing the SAFE-6G implementation. Instead, it requires separating the common lifecycle from the trustworthiness-specific models, policies, and enforcement entities.

This distinction is important for the long-term exploitation of CoCo. The common coordination logic can remain stable while profile-specific components are selected according to the management domain. A trustworthiness profile can continue to use the five SAFE-6G Trust Functions, whereas a RAN profile could use radio controllers and RAN-specific AI models, a 5G Core profile could use network-function and exposure APIs, and a service or edge profile could use service orchestration and placement mechanisms. The same intent-management lifecycle can therefore be reused while the underlying cognitive model and action interfaces vary by profile.

5.1.2 CoCo AS A 3GPP-ALIGNED INTENT HANDLING FUNCTION

3GPP TS 28.312 defines intent-driven management by allowing an MnS consumer to express expectations, requirements, goals, and constraints without prescribing how the underlying system should realize them [11]. The specification introduces the Intent Handling Function (IHF) as the logical function that provides intent-driven management capabilities and structures intent information

through concepts such as intent expectations, expectation objects, expectation targets, and contexts. This “what rather than how” separation is particularly relevant to 6G environments in which the same high-level requirement may be fulfilled through different combinations of radio, core, edge, cloud, and service-layer actions. The generic management-service framework and service-exposure principles are further specified in 3GPP TS 28.532 [12].

From this perspective, CoCo can be positioned as the cognitive realization of an IHF on the MnS producer side. CoCo does not replace the existing management systems responsible for executing network operations. Rather, it provides the semantic, reasoning, and coordination layer that receives a desired state, interprets the associated expectations and targets, evaluates feasibility and conflicts, selects the appropriate domain-specific mechanisms, coordinates execution, and reports the resulting fulfilment status. Figure 32 illustrates this future positioning of CoCo between intent consumers and heterogeneous management domains.

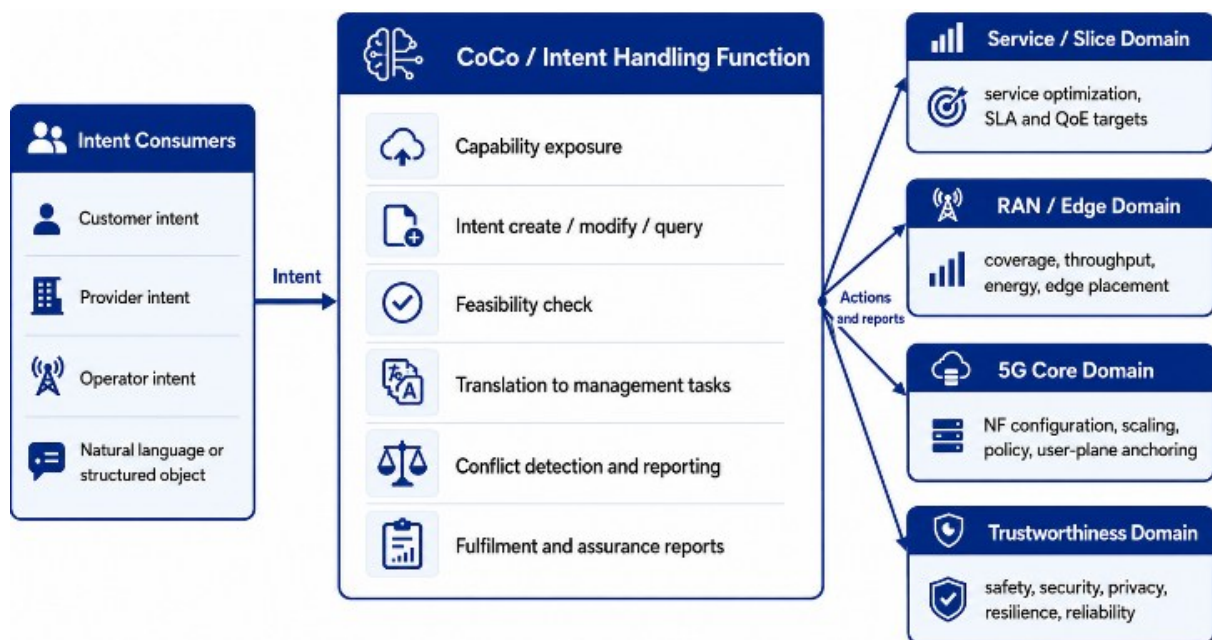


Figure 32: Future positioning of CoCo as a 3GPP-aligned Intent Handling Function. The trustworthiness functionality developed in SAFE-6G represents one supported management profile alongside other service, RAN/edge, and 5G Core domains (adapted from [13])

In the SAFE-6G deployment, the human user remains the originator of the requirement, while the Chatbot acts as the user-facing intent-acquisition mechanism that produces the structured input consumed by CoCo. In a broader 3GPP-oriented deployment, the MnS consumer role may instead be assumed by an OSS, orchestrator, management application, or another automated management function. Consequently, the Chatbot should be regarded as one possible northbound front end rather than as an architectural prerequisite for CoCo. This separation preserves the human-centric interaction demonstrated in SAFE-6G while allowing the same CoCo component to participate in operator-grade management architectures.

5.1.3 PROFILE-DRIVEN ARCHITECTURE AND MAPPING TO THE SAFE-6G IMPLEMENTATION

A profile-driven architecture provides the mechanism for separating the reusable CoCo lifecycle from domain-specific intelligence and enforcement. An intent profile defines the supported intent family, applicable expectation objects and targets, required contextual information, cognitive model, reasoning strategy, and action adapters through which the resulting decision can be enforced. Figure 33 illustrates this generic organization. Intent ingestion and normalization, lifecycle management, reasoning, and reporting remain common, while the cognitive model and action adapters are selected according to the requested management domain.

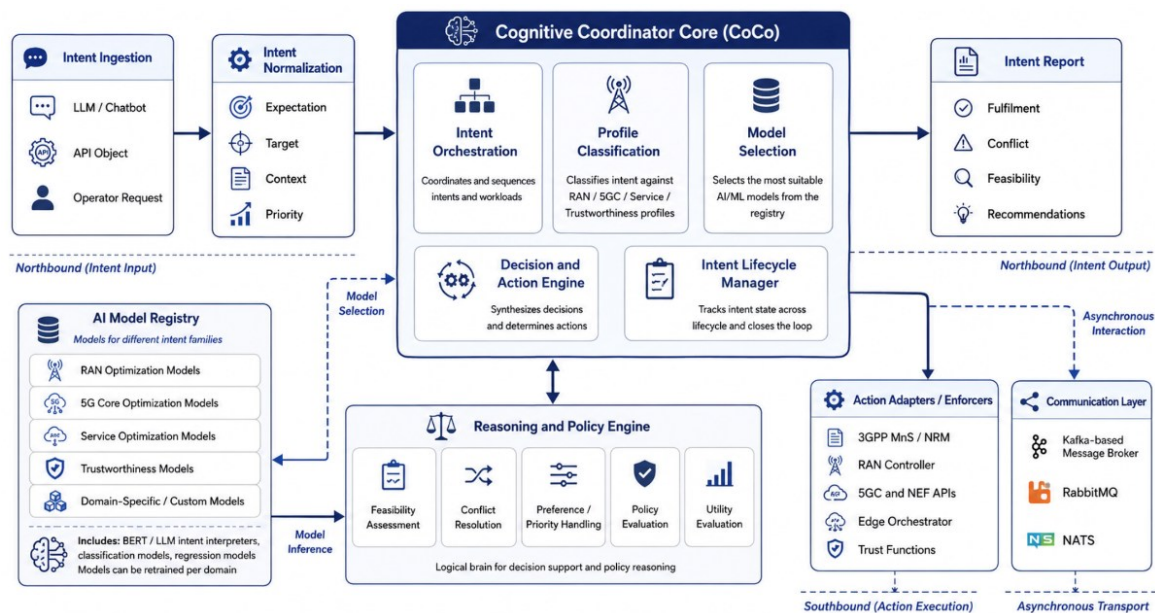


Figure 33: Profile-driven CoCo architecture. Common intent lifecycle and reporting functions are decoupled from profile-specific cognitive models, reasoning policies, communication mechanisms, and Action Adapters (adapted from [5])

The correspondence between the implementation documented in this deliverable, the generic CoCo abstraction, and the principal concepts of 3GPP intent-driven management is summarized in Table 13. This mapping is architectural rather than a claim of complete standards conformance; it identifies how the functionality already implemented within SAFE-6G can provide the foundation for a standards-aligned evolution.

Under this abstraction, the SAFE-6G trustworthiness implementation becomes the CoCo Trustworthiness Profile rather than the architectural boundary of CoCo. Its cognitive model maps user trust requirements onto the five trustworthiness dimensions; the Trust Functions provide the corresponding domain-specific assessment and enforcement capabilities; and the Conflict Resolution Manager produces a feasible calibrated configuration.

Current SAFE-6G implementation	Generic CoCo abstraction	3GPP intent-management relation
SAFE-6G Chatbot + /submit API	Intent ingestion / northbound front end	Intent submission by an MnS consumer or user-facing application

Structured Chatbot payload and CoCo preprocessing	Intent normalization	Intent expectations, expectation objects, targets, contexts, and priorities
CAIO	Intent orchestration and lifecycle management	Coordination and tracking of the intent-handling lifecycle
Machine Learning Module	Cognitive Model Registry / model inference	Profile-specific interpretation of intent targets and desired outcomes
User Registry + Digital Twin + TF feedback	Context and feasibility information	Context handling and feasibility assessment
Knowledge Base + CRM	Reasoning and Policy Engine	Conflict detection, resolution, policy, and priority handling
Kafka Message Broker	Communication Layer	Asynchronous exchange with domain-specific functions
SAFE-6G Trust Functions	Trustworthiness Action Adapters / Enforcers	Domain-specific feasibility assessment and fulfilment actions
CAIO state + API response + GUI	Intent reporting and observability	Fulfilment, feasibility, conflict, and execution-status reporting
XAI Aggregator	Complementary explainability capability	Transparency for AI-supported decisions; complementary to the core TS 28.312 mapping

Table 13: Mapping between the current SAFE-6G CoCo implementation, the generic CoCo abstraction, and 3GPP intent-driven management concepts

Other profiles can reuse the same lifecycle while introducing different models and adapters. Examples include service and slice optimization, RAN coverage and energy optimization, 5G Core configuration and scaling, and edge/cloud-continuum placement or migration. This profile-based approach is consistent with the generic architecture proposed in the accompanying CoCo research work [13].

5.1.4 STANDARDS-ORIENTED FUTURE ACTIVITIES

The current SAFE-6G implementation should therefore be regarded as a functional foundation for, rather than a complete realization of, the 3GPP intent-management framework. Future development can progressively strengthen the alignment with TS 28.312 and the generic management-service principles of TS 28.532 through the following activities:

- **Intent information-model alignment.** Represent intent expectations, expectation objects, expectation targets, contexts, preferences, and priorities explicitly within the CoCo northbound information model.
- **Capability exposure.** Expose the intent families, managed objects, targets, cognitive models, and action capabilities supported by each CoCo profile so that an MnS consumer can determine what the IHF is able to process before submitting an intent.
- **Intent lifecycle operations.** Extend the current request-oriented API towards standardized operations for creating, modifying, activating, deactivating, querying, and deleting intents, while maintaining explicit lifecycle state.
- **Generic feasibility and conflict handling.** Abstract the current Digital Twin, Trust Function feedback, Knowledge Base, and CRM mechanisms into profile-independent interfaces for feasibility assessment, conflict detection, negotiation, and resolution.
- **Structured fulfilment and assurance reporting.** Generalize the CAIO session state and response mechanisms into report objects that maintain traceability between the original

intent, feasibility and conflict decisions, executed management actions, and achieved outcomes.

- **Southbound Action Adapter abstraction.** Decouple the current trustworthiness-specific Kafka and Trust Function interactions from the common CoCo lifecycle by introducing adapters for RAN controllers, 5G Core and NEF APIs, service orchestrators, edge/cloud platforms, OpenCAPIF-enabled capabilities, and other programmable management interfaces.
- **Multi-profile and multi-network validation.** Validate the generic CoCo architecture beyond trustworthiness through additional service, RAN, Core, and edge/cloud intent profiles and assess operation across heterogeneous management and network domains.

This standards-oriented evolution should not be interpreted as a claim that the current SAFE-6G implementation already conforms to every information-model, interface, and procedure defined by TS 28.312. Rather, the implementation already realizes the key architectural pattern: high-level requirements are transformed into machine-actionable objectives, evaluated against contextual information and domain capabilities, reconciled when conflicts arise, translated into management actions, and monitored through execution outcomes and reports. Future work will formalize these abstractions, extend the northbound and southbound interfaces, and validate the resulting generic CoCo component across multiple intent profiles and network-management domains.

The SAFE-6G implementation documented in this deliverable can therefore be considered both a concrete trustworthiness solution and the first operational validation environment for a broader CoCo concept. By preserving the implemented cognitive coordination mechanisms while decoupling them from trustworthiness-specific semantics, CoCo can evolve into a reusable, open, and extensible building block for 3GPP-aligned intent-driven management in B5G/6G networks.

6 CONCLUSION

This deliverable marks the transition of the SAFE-6G trustworthiness framework from the architectural definitions established in D4.1 to the development and deployment of the Cognitive Coordinator (CoCo) and the XAI Aggregator as operational components of the SAFE-6G platform. Through their integration with the five modular Trust Functions and the broader SAFE-6G ecosystem, CoCo and the XAI Aggregator provide the mechanisms required to translate user requirements into coordinated, adaptive, and explainable trustworthiness decisions.

During their development, CoCo and the XAI Aggregator have evolved beyond their initial role of supporting transparent trustworthiness provisioning to address broader operational challenges within SAFE-6G. These include multi-user and multi-tenant operation, differentiated user roles and trustworthiness privileges, inspection of the decisions and processes involved in trustworthiness management, and dynamic adaptation to changing user, network, and operational conditions. This evolution also reflects the distributed nature of SAFE-6G, in which the individual Trust Functions are implemented as distinct functional entities rather than as parts of a monolithic component. CoCo and the XAI Aggregator therefore bridge these distributed components and enable them to operate as part of a coherent end-to-end framework. In particular, the CAIO has evolved from a sequential orchestrator into a dynamic, context-aware lifecycle controller complemented by a dedicated GUI-based inspection layer.

The implementation choices, internal developments, functionality, and deployment approaches documented in this deliverable therefore extend beyond the technical realization of the architecture introduced in D4.1. They establish the mechanisms required to coordinate heterogeneous SAFE-6G components, accommodate different users and operational contexts, and make trustworthiness decisions accessible for inspection and explanation. In doing so, CoCo and the XAI Aggregator provide the technical foundation for validation of the SAFE-6G trustworthiness framework within the project's use cases and experimentation environments, under diverse threat scenarios, trustworthiness configurations, user requirements, and network conditions. At the same time, the implemented lifecycle provides a practical basis for extending CoCo beyond trustworthiness towards a generic, profile-driven intent-handling component, as discussed in the following subsection.

7 REFERENCES

- [1] FastAPI. [Online]. Available: <https://fastapi.tiangolo.com/>.
- [2] Pydantic. [Online]. Available: <https://pypi.org/project/pydantic/>.
- [3] A. Essakine, Y. Cheng, C.-W. Cheng, L. Zhang, Z. Deng, L. Zhu, C.-B. Schönlieb and A. I. Aviles-Rivero, “Where Do We Stand with Implicit Neural Representations? A Technical and Performance Survey”.
- [4] V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell and G. Wetzstein, “Implicit Neural Representations with Periodic Activation Functions”.
- [5] BERT. [Online]. Available: https://huggingface.co/docs/transformers/en/model_doc/bert.
- [6] pytorch. [Online]. Available: <https://github.com/pytorch/pytorch>.
- [7] Transformers. [Online]. Available: <https://huggingface.co/docs/transformers/en/index>.
- [8] Zenodo. [Online]. Available: <https://zenodo.org/records/15512671>.
- [9] NLTK. [Online]. Available: <https://www.nltk.org/howto/wordnet.html>.
- [10] Adam. [Online]. Available: <https://docs.pytorch.org/docs/2.14/generated/torch.optim.AdamW.html>.
- [11] 3. T. 2. V19.5.0, ““Management and orchestration; Intent driven management services for mobile networks (Release 19),” 3rd Generation Partnership Project, Mar. 2026.”.
- [12] 3. T. 28.532, ““Management and orchestration; Generic management services,” 3rd Generation Partnership Project.”.
- [13] D. Roidis, S. Georgoulas, E. Anagnostopoulos, H. Koumaras, V. Pitsilis, G. Makropoulos, I. Kontizas and G. Koumaras, ““CoCo: An Open-Source Cognitive Coordinator for 3GPP-Aligned Intent-Driven Management in 6G Networks,” accepted for publication in Proc. IEEE CloudNet, 2026.”.