



AI-DRIVEN CYBERSECURITY FOR FINANCIAL SERVICE PROVIDERS

D2.5 – LLM-BASED CYBERSECURITY ORCHESTRATION DESIGN

Lead Beneficiary	INNOV-ACTS Limited (INNOV)
Work Package Ref.	WP2-LLM-Powered Reference Architecture, Specification and Systems Integration
Task Ref.	T2.4-LLM Cybersecurity Orchestrator Specifications
Deliverable Title	LLM-based Cybersecurity Orchestration Design
Due Date	30/09/2026
Delivered Date	29/09/2026
Revision Number	V1.0
Dissemination Level	PU – Public
Type	R — Document, report
Document Status	Final
Review Status	Reviewed
Document Acceptance	Acceptable
EC Project Officer	Rodica MORARIU

DIGITAL-ECCC-2024-DEPLOY-CYBER-07-KEYTECH



Co-funded by
the European Union



ECCC
EUROPEAN CYBERSECURITY
COMPETENCE CENTRE

The project funded under Grant Agreement No 101249596 is supported by the European Cybersecurity Competence Centre

DISCLAIMER

The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the European Union. Neither the ECCC nor the European Commission is responsible for any use that may be made of the information contained therein.

COPYRIGHT MESSAGE

This report, if not confidential, is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0); a copy is available here: <https://creativecommons.org/licenses/by/4.0/>. You are free to share (copy and redistribute the material in any medium or format) and adapt (remix, transform, and build upon the material for any purpose, even commercially) under the following terms: (i) attribution (you must give appropriate credit, provide a link to the license, and indicate if changes were made; you may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use); (ii) no additional restrictions (you may not apply legal terms or technological measures that legally restrict others from doing anything the license permits).

CONTRIBUTING PARTNERS

Partner Acronym	Role ¹	Name Surname ²
INNOV	Lead Beneficiary	George Fatouros, Despina Tomkou, John Soldatos
UBI	Contributor	Giannis Ledakis, Nikos Papageorgopoulos
IQB	Contributor	Cosmina Stalidi
UNP	Contributor	Ricardo Ladeira
UPRC	Contributor	Konstantinos Mavrogiorgos
K3Y	Contributor	Panagiotis Radoglou Grammatikis, Evangelia Pantraki, Christos Dalamangas, Nikolaos Ntampakis, Dimitrios Balompas, Antonios Sarigiannidis
UNP	Internal Reviewer	Gonçalo Rodrigues
UPRC	Internal Reviewer	Georgios Makridis
UBI, GFT	Quality Assurance	Louiza Kachrimani, Michele Canteri

REVISION HISTORY

Version	Date	Partner(s)	Description
v0.1	10/07/2026	INNOV	ToC Version
v0.2	05/08/2026	INNOV	Content added for Sections 4–6
v0.3	10/08/2026	INNOV, UPRC	Content added for Sections 7–13
v0.4	26/08/2026	INNOV	Section 2 added
v0.5	27/08/2026	INNOV, UBI	Section 3 added; Section 8 extended
v0.6	28/08/2026	UBI	Architectural figures added and related text revised
v0.7	31/08/2026	INNOV	Section 8 revised, Section 1 drafted
v0.8	02/09/2026	INNOV	Executive Summary added
v0.9	09/09/2026	IQB	Updated Section 4 and 7
v0.10	10/09/2026	UPRC, UNP	Updated Section 8
v0.11	14/09/2026	INNOV	Editorial pass completed
v0.12	21/09/2026	UNP, UPRC	Internal peer review
v0.13	23/09/2026	INNOV	Revised after internal peer review
v1.0	29/09/2026	UBI	Final quality check and submission to the EU

¹ Lead Beneficiary, Contributor, Internal Reviewer, Quality Assurance

² Can be left void

LIST OF ABBREVIATIONS

Acronym	Definition
AI	Artificial Intelligence
API	Application Programming Interface
CRA	CyberAid Coordinator / Central Router Agent
CTI	Cyber Threat Intelligence
DORA	Digital Operational Resilience Act
EU AI Act	European Union Artificial Intelligence Act
GDPR	General Data Protection Regulation
HITL	Human-in-the-Loop
JSON	JavaScript Object Notation
LLM	Large Language Model
MCP	Model Context Protocol
NFR	Non-Functional Requirement
NIS2	Network and Information Security Directive 2
RAG	Retrieval-Augmented Generation
SIEM	Security Information and Event Management
SKM	Security Knowledge Model
SOC	Security Operations Centre
STIX/TAXII	Structured Threat Information Expression / Trusted Automated Exchange of Intelligence Information
WP	Work Package
XDR	Extended Detection and Response

EXECUTIVE SUMMARY

D2.5 is the output of Task 2.4 and specifies the design of the CyberAid LLM orchestration layer: the coordination logic and shared agent runtime that sit between the alerting and integration fabric defined in D2.2 and the specialist agents to be built under WP3, WP4 and WP5. Its purpose is narrow and practical. Six technical tasks are building LLM-based cybersecurity agents in parallel, and without a shared architecture each would derive its own approach to authorisation, state, audit and output format. This deliverable fixes that architecture once, to a level of detail sufficient for T5.1 and the agent-building tasks to build against a common contract rather than separate ones.

The design rests on a systematic review of 59 primary studies of LLM-based agents in cybersecurity, conducted within the project under PRISMA methodology, together with an architecture paper the project presented at DCOSS-IoT 2026. Two findings from that review drove the two central decisions in this deliverable. Centralised orchestration is the only pattern the review credits with structural auditability, a property a DORA-regulated deployment cannot trade away, so it is adopted as the platform-level pattern, with its known bottleneck weakness mitigated by a direct-invocation path rather than by abandoning it. And of the 59 studies, 52 run fully automated and only one uses human approval, which is not a posture this design can inherit for a regulated financial deployment; the tiered human-in-the-loop model is accordingly treated as a first-class design element.

Every agent in the platform, the coordinator and every specialist, is built from one canonical execution pattern rather than separate bespoke ones (Section 4). CyberAid-owned Runtime Core code holds enforcement, audit, finding validation and human-in-the-loop routing regardless of which agent is running; the orchestration engine beneath it, proposed as LangGraph, does only execution and state persistence. This separation keeps enforcement independent of the orchestration engine and of any individual agent's instructions. The Security Context, immutable for the life of a workflow, denying by default rather than falling back to a permissive baseline, and enforced centrally rather than per agent, is the schema that makes this enforceable (Section 6.1). A guardrails framework at agent boundaries complements it, screening injected instructions on input and requiring grounded evidence, a classification ceiling and calibrated confidence on output.

Work reaches an agent through coordinator-mediated dispatch by default, or, where an alert's classification is unambiguous, through direct specialist invocation that skips coordination without skipping enforcement (Section 3.3). Where two specialists disagree about the same event, the design escalates the disagreement to a human rather than resolving it by averaging confidence scores or discarding the less severe reading (Section 9.3). Everything an agent produces is a structured finding: an attributed record of what was examined, which tools and model produced it, how confident it is, and what review it requires, validated through a single gate before it can be acted on (Section 10.1). The Reporting agent composes findings into five report types for readers who will not read findings directly, and asserts only what its findings support.

Three design properties carry most of the regulatory weight: attribution of every finding to its evidence, tools and model; oversight proportionate to the reversibility of the action rather than uniform review; and data movement bounded by reference rather than by copy. Together they answer DORA's incident-documentation obligations, NIS2's notification timelines, the EU AI Act's human-oversight requirements for high-risk cybersecurity tooling, and GDPR's erasure obligations (Section 11.2), stopping short of a legal compliance determination, which rests with the deploying institutions. Performance targets are traced individually to D2.1's non-functional requirements (Section 11.1). The proposed technology stack, LangGraph, FastAPI with Pydantic, SQLAlchemy, and Kafka for the alerting bridge, is chosen for on-premises deployability and independence from any single model provider (Section 7).

What this deliverable fixes applies to every agent regardless of domain: the canonical architecture, the Security Context, the finding schema and the review tiers. What it leaves open is deliberate. Each specialist's internal reasoning and tool bindings belong to the task that owns it, and three storage decisions and several agent realisation patterns remain work in progress (Section 8). Section 12 fixes in advance the criteria an implementation will be measured against. The claim this design makes is a modest one. It does not make an

agent's judgement correct, but it makes that judgement inspectable, so that when an agent is wrong, and in this domain it will be some proportion of the time, the error is visible, attributable and correctable rather than an unexplained output an analyst has to trust or discard.

TABLE OF CONTENTS

1.	Introduction	10
1.1	Objectives of the Deliverable	10
1.2	Insights from Other Tasks and Deliverables	10
1.3	Structure	11
2.	Background and State of the Art	11
2.1	LLM Agent Concepts	11
2.2	Systematic Review Methodology and Scope	12
2.3	Taxonomy of Orchestration Patterns	13
2.4	Coverage Gaps and Open Challenges Relevant to CyberAId	14
2.5	Positioning of the CyberAId Design Against the Review	15
3.	Architectural Positioning of the LLM Orchestration Layer	15
3.1	Position within the CyberAId Reference Architecture	16
3.2	The Alerting Layer as the SIEM/Tooling Bridge	17
3.3	Orchestration Entry Points	18
4.	The Canonical LLM Agent Architecture	18
4.1	Runtime Core vs. Orchestration Engine	19
4.2	Shared Agent Contract	19
4.3	Tool Integration	20
4.4	Knowledge Access	20
4.5	Observability and Auditing	20
5.	State Handling, Memory and Shared Data Infrastructure	21
5.1	Workflow State and Checkpointing	21
5.2	Context Policy: Hot, Warm and Cold Tiers	22
5.3	Organisation Context via CYBERAID.md	23
5.4	Scoped and Longer-Term Memory	23
5.5	Shared Knowledge Infrastructure and Common Tooling	23
6.	Security Context and Governance Model	24
6.1	Security Context Schema and Enforcement Points	25
6.2	Guardrails Framework	26
7.	Proposed Technology Stack for the Orchestration Layer	27
7.1	Orchestration Engine and Backend Framework	28
7.2	Model Gateway and Provider Registry	28
7.3	Persistence and Knowledge Stores	29
7.4	Tool Integration Technologies	29
7.5	Alerting/Event Bridge and Operator Interfaces	30

8. Agent Specifications 30

8.1 CyberAId Coordinator (Central Router Agent) 31

8.2 Specialist Agents Overview 31

8.3 Agent Realisation Patterns 32

8.4 Agent-to-Task Mapping.....32

9. Orchestration Logic, Workflows and Human-in-the-Loop Design.....34

9.1 Orchestration Patterns in Use 34

9.2 Workflow Type Catalogue and Dispatch Logic.....35

9.3 Result Aggregation and Conflict Resolution 35

9.4 HITL Interaction Model.....36

10. Structured Findings and Reporting.....36

10.1 Finding Schema Specification.....37

10.2 Report Types and Composition Logic.....38

11. Non-Functional Requirements and Regulatory Alignment.....38

11.1 Performance Requirements for the Orchestration Layer 38

11.2 Regulatory Alignment of Design Choices.....40

12. Validation Approach 40

13. Conclusions and Next Steps 41

LIST OF FIGURES

Figure 1. Relationship of D2.5 to Other CyberAId Tasks and Deliverables 10

Figure 2: Taxonomy of LLM agent architectural patterns 12

Figure 3: PRISMA 2020 flow diagram for the systematic literature review. 13

Figure 4: CyberAId reference architecture and the LLM/Agentic layers.....16

Figure 5: The canonical LLM agent with one execution pattern, shared by the coordinator and every specialist.....18

Figure 6: Security Context schema, enforcement points, and audit flow 25

LIST OF TABLES

Table 1: Multi-agent orchestration patterns and their trade-offs 13

Table 2: Review directions addressed by this deliverable 15

Table 3: Relationship between the D2.2 architecture layers and this deliverable 16

Table 4: Separation of concerns between the Runtime Core and the orchestration engine 19

Table 5: Audit event categories recorded by the runtime 21

Table 6: Workflow state fields.....21

Table 7: Context tiers.....22

Table 8: Open technology options for the shared data infrastructure 24

Table 9: Security Context fields.....25

Table 10: Enforcement points and the invariants checked at each.....26

Table 11: Division of responsibility between enforcement and guardrails 26

Table 12: Proposed technologies by layer.....28

Table 13: Model provider classes supported by the gateway 28

Table 14: Agent roster and workflow types 30

Table 15: Agent roles mapped to implementing tasks and partners 32

Table 16: Workflow type catalogue 35

Table 17: Human-in-the-loop tiers 36

Table 18: Finding schema field groups 37

Table 19: Performance and deployment requirements bearing on the orchestration layer 38

Table 20: Architecture requirements from D2.1 specified by this design 39

Table 21: Design responses to regulatory obligations 40

Table 22: Evaluation criteria and pass thresholds 40

1. Introduction

D2.5 documents the outputs of the Task 2.4 presenting the design specification for the CyberAid LLM orchestration layer. It defines the coordination logic and shared agent runtime that sit between the alerting and integration fabric specified in D2.2 and the specialist agents built under WP3, WP4 and WP5.

1.1 Objectives of the Deliverable

The purpose of this deliverable is to specify the design of the CyberAid LLM orchestration layer to a level of detail sufficient for T5.1 and the other agent-building tasks across WP3-5 to build against one shared architecture rather than each deriving its own. It defines the canonical agent architecture common to the coordinator and every specialist (Section 4), the shared state and context policy (Section 5), the Security Context and governance model (Section 6), the finding schema and reporting contract (Section 10), and the tiered human-in-the-loop model that gates high-impact action (Section 9.4). These elements are specified as contracts an implementation must satisfy.

D2.5 documents design, not implementation. The internal reasoning, tool bindings and domain logic of each specialist agent are left to the task that owns it, while this deliverable states the domain and scope boundary of each agent, not how it reasons.

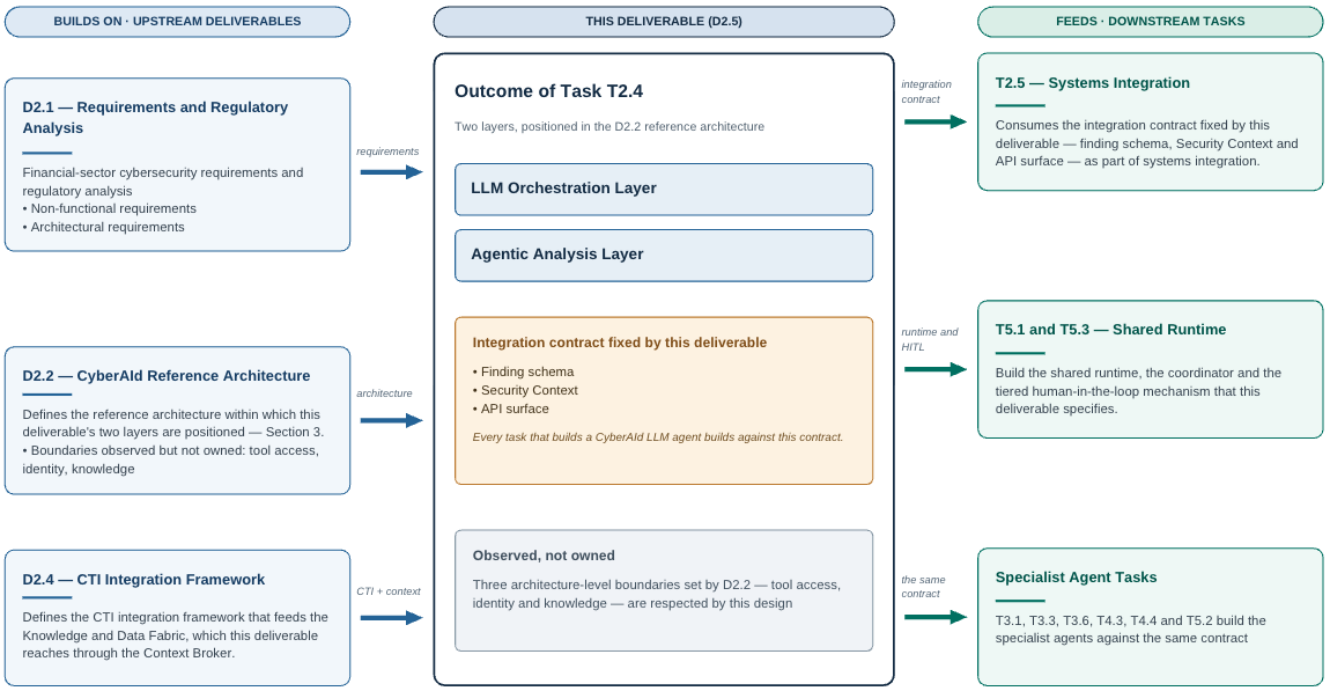


Figure 1: Relationship of D2.5 to Other CyberAid Tasks and Deliverables

1.2 Insights from Other Tasks and Deliverables

As illustrated at Figure 1, this deliverable is built to satisfy requirements fixed by D2.1, to sit on the architecture defined by D2.2, and to consume the CTI integration framework defined by D2.4. D2.1 establishes the financial-sector cybersecurity requirements and regulatory analysis this deliverable answers. Its non-functional requirements are traced individually against the design in Section 11.1, and its architectural requirements, among them dual-mode agent invocation, per-agent knowledge partitions, the tool adapter layer, and hallucination mitigation are specified directly by Sections 3 to 10. D2.2 defines the CyberAid reference architecture within which this deliverable's two layers, the LLM Orchestration layer and the Agentic Analysis layer, are positioned in Section 3, which also states the three architecture-level

boundaries (tool access, identity, knowledge) that this design observes but does not own. D2.4 defines the CTI integration framework that feeds the Knowledge and Data Fabric this deliverable reaches through the Context Broker.

This deliverable's outputs feed every task building an LLM agent for CyberAId. T2.5 consumes the integration contract this deliverable fixes, namely the finding schema (Section 10.1), the Security Context (Section 6.1) and the API surface (Section 7.5), as part of systems integration. T5.1 and T5.3 build the shared runtime, coordinator and tiered human-in-the-loop mechanism this deliverable specifies. T3.1, T3.3, T3.6, T4.3, T4.4 and T5.2 build the specialist agents against the same contract, as mapped individually in Section 8.4.

1.3 Structure

Section 2 sets out the evidence base for the design grounded on the systematic review of LLM-based agents in cybersecurity conducted within the project, the taxonomy of orchestration patterns and coverage gaps it establishes, and how this deliverable positions itself against the review's open research directions. Section 3 locates the LLM orchestration layer within the CyberAId reference architecture, specifies the alerting layer as the bridge between the SIEM/XDR tooling and the orchestration layer, and defines the entry points through which work reaches an LLM agent. Section 4 specifies the canonical LLM agent architecture shared by the coordinator and every specialist, and the separation of concerns between the CyberAId-owned Runtime Core and the orchestration engine beneath it. Section 5 specifies workflow state and checkpointing, the context policy and scoped memory. Section 6 specifies the Security Context schema and its enforcement points, and the guardrails framework that complements enforcement at agent boundaries.

Section 7 proposes the technology stack for the orchestration layer, covering the orchestration engine and backend framework, the model gateway and provider registry, persistence and knowledge stores, tool integration technologies, and the alerting bridge and operator interfaces. Section 8 specifies the agent roster including the CyberAId Coordinator, the specialist agents and their domains as well as the realisation patterns available to building them. Section 9 specifies the orchestration logic, the available workflows and dispatch rules, result aggregation and conflict resolution, and the human-in-the-loop interaction model. Section 10 specifies the finding schema and the report types composed from it. Section 11 traces performance requirements to D2.1's requirements and maps design decisions to the regulatory obligations of DORA, NIS2, GDPR and the EU AI Act. Section 12 defines the evaluation criteria and pass thresholds against which an implementation of this design will be measured and finally Section 13 concludes the deliverable.

2. Background and State of the Art

The design in this deliverable rests on a peer-reviewed systematic review of LLM-based agents in cybersecurity conducted within the project [1], together with an architecture paper presented at DCOSS-IoT 2026 [2]. This section summarises what that evidence base establishes and what it leaves open and positions the CyberAId orchestration design against both.

2.1 LLM Agent Concepts

An LLM agent differs from a conventional model deployment in that it operates within an action loop, as in the tool-calling (SA-TC) pattern of Figure 2. The agent perceives context, reasons over available tools, issues an action, observes the outcome, and iterates towards a goal [3]. A model asked to classify an email is not an agent; a component that retrieves the message, queries a reputation service, checks a sender history and then produces a judgement is.

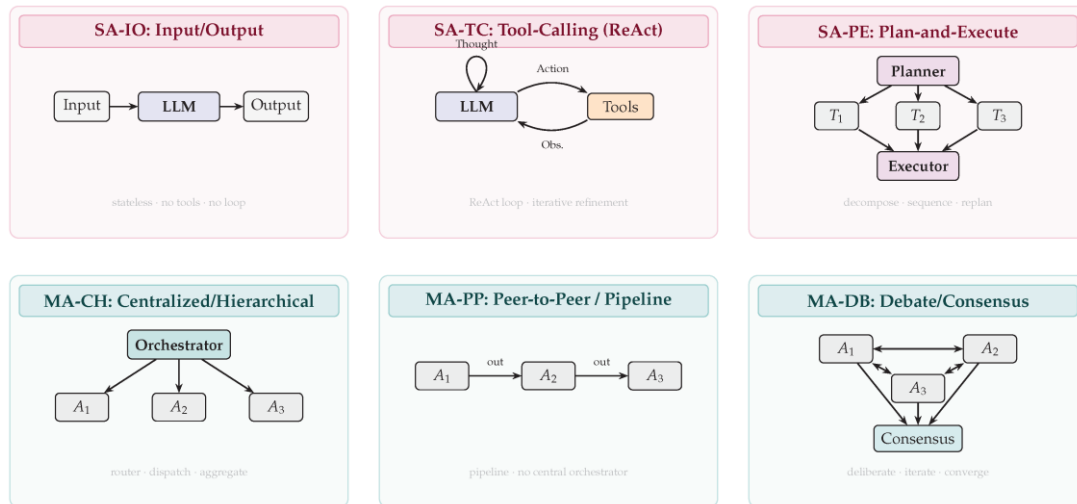


Figure 2: Taxonomy of LLM agent architectural patterns

Tools are the typed operations an agent may invoke such as a query against a monitoring system, a file read, a lookup against threat intelligence. The agent selects from a declared set; the runtime executes and returns a structured result. Agent context is the working information available to the model for a given step, bounded by the model’s context window, which is a hard limit that long-running security analyses exceed routinely.

The Model Context Protocol (MCP) [4] standardises how an agent reaches external tools or services like a SIEM, so a capability exposed once can be consumed by any compliant agent without bespoke integration. Retrieval-Augmented Generation (RAG) [5] supplies an agent with material retrieved at inference time from an external store rather than relying on parametric knowledge; the store is typically a vector database, which indexes text by semantic embedding so that retrieval returns passages related in meaning rather than matched by keyword.

2.2 Systematic Review Methodology and Scope

The review [1] follows PRISMA guidelines. Records were identified through 59 structured queries resolving predominantly to arXiv, Semantic Scholar, the ACM Digital Library, IEEE Xplore, USENIX, MDPI, SpringerLink and Elsevier ScienceDirect, supplemented by citation chaining, covering January 2022 to April 2026. As shown at Figure 3, after structured inclusion and exclusion screening, 59 primary studies were classified along five dimensions: security function, agent architecture pattern, knowledge augmentation strategy, human-in-the-loop posture, and evaluation rigor.

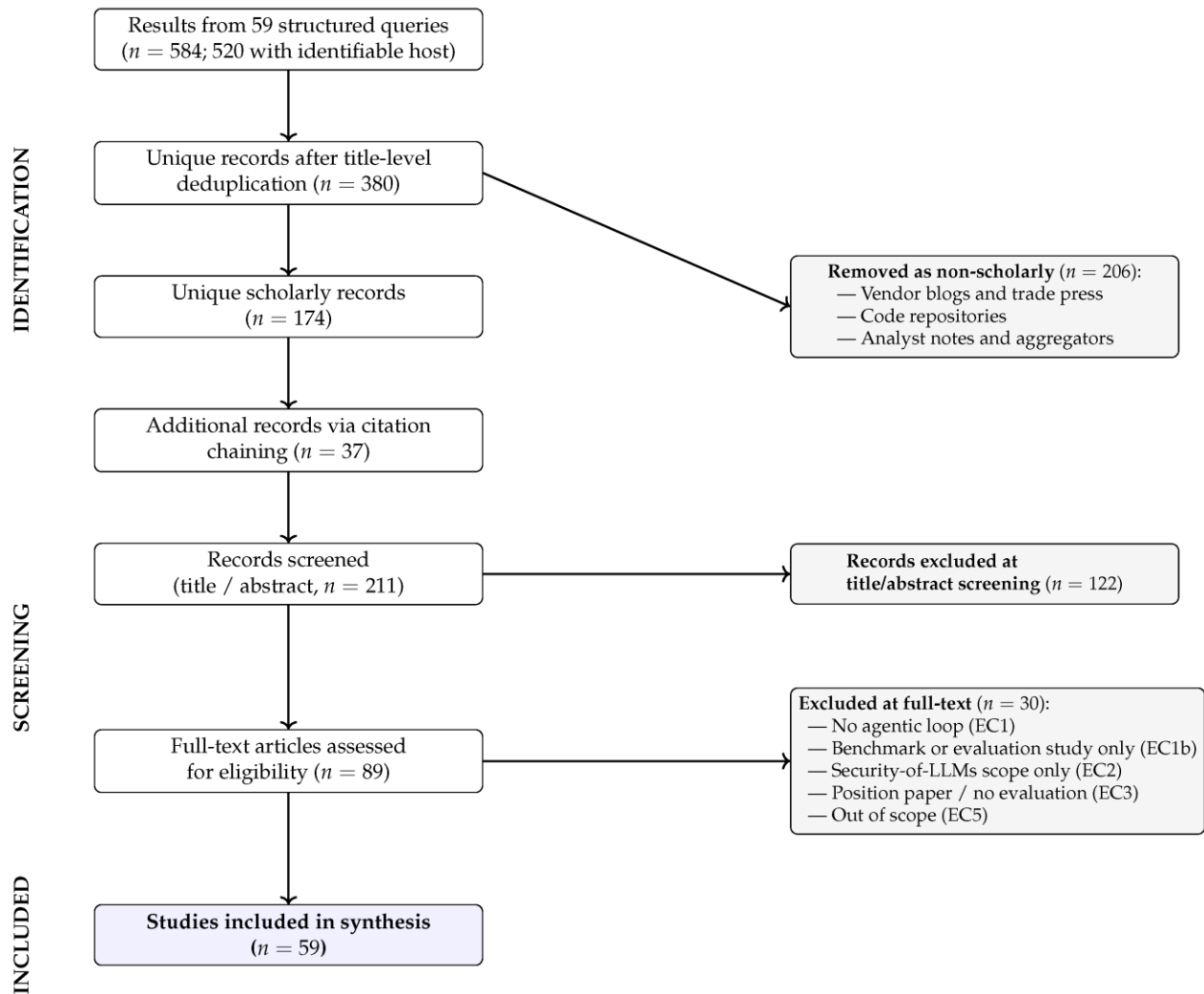


Figure 3: PRISMA 2020 flow diagram for the systematic literature review.

Two scope qualifications matter for how the findings below should be read. An independent verification search across six bibliographic databases recovered 86.3% of the studies the primary search had surfaced while indicating a total eligible literature of roughly 400 works, so the corpus is reported as a documented subset rather than an exhaustive census. And evaluation rigor is coded on a four-point scale from conceptual (E1) through public benchmark (E2) and controlled testbed (E3) to real-world deployment (E4).

2.3 Taxonomy of Orchestration Patterns

The review codes six architecture patterns, three single-agent and three multi-agent. Single-agent tool-calling remains the most prevalent at 30.5% of studies, while centralised multi-agent orchestration, present in 18.6%, produces the strongest reported performance gains — prevalence and performance point in opposite directions.

Twenty-two studies use multi-agent architectures, distributed across three patterns (see Table 1).

Table 1: Multi-agent orchestration patterns and their trade-offs

Pattern	Count	Best reported result	Key trade-off
Centralised / hierarchical (MA-CH)	11	4.3× over single-agent on zero-day exploitation	Orchestrator bottleneck and single point of failure; strong audit trail

Pattern	Count	Best reported result	Key trade-off
Pipeline / parallel (MA-PP)	8	100% actionable incident response rate versus 1.7% single-agent	Phase-coupling brittleness; summariser modules needed to preserve context
Debate / consensus (MA-DB)	3	False-positive rate reduced from over 30% to under 7%	Latency overhead; convergence not guaranteed; strongest reliability gains

Three observations from this table bear directly on the present design.

The audit advantage of centralised orchestration is structural rather than incidental: an orchestrator that dispatches work necessarily holds a complete record of which agents were activated, on what input, with what result and confidence which is the material DORA incident documentation requires [6].

The pipeline result is a finding about orchestration, not scale. MyAntFarm.ai achieved a 100% actionable response rate across 348 trials using identical TinyLlama-1B models arranged in a pipeline, against 1.7% for the same model deployed as a single agent [7]. Orchestration design, not model size, was the determinant which matters directly for institutions constrained to on-premises inference. The strongest centralised result, a 4.3× improvement on zero-day exploitation [8], comes from parallel dispatch to specialists rather than from a larger model.

The debate pattern’s reliability gains come with a caution. Audit-LLM’s own ablation [9] attributes only a modest share of its false-positive reduction to the debate step specifically, and an incident-response study reported in the review finds that adding structural diversity does not reliably outperform a simpler, clearly-led homogeneous team [1]. More elaborate coordination is not automatically better.

2.4 Coverage Gaps and Open Challenges Relevant to CyberAid

Four findings from the review define the space this deliverable operates in.

Coverage is heavily skewed towards offensive security. Penetration testing alone accounts for 50.8% of the corpus. Incident response and digital forensics together account for five studies, and no study addresses systematic digital forensics beyond log retrieval. Compliance and risk assessment accounts for exactly one study which is also the corpus’s only banking-specific system [10]. The domains CyberAid’s pilots most need are the ones the literature has least examined.

Human oversight is close to absent. Of the 59 studies, 52 adopt fully automated postures. Six are human-guided, and all six sit within penetration testing. One study — a SIEM-integrated intrusion detection system — uses human approval. No study in the corpus is coded human-overseen. Defensive systems are more automated than offensive ones, which inverts the intuition that higher-stakes defensive work would retain more human involvement.

No study reaches production-grade evaluation. The entire field currently rests on controlled laboratory assessment. Reported performance figures should be read accordingly.

The regulatory gap is documented rather than closed. DORA requires an initial notification within four hours of classifying a major incident, an intermediate report within 72 hours and a final report within one month [6]; NIS2 requires a 24-hour early warning and 72-hour full notification [11]; and the EU AI Act classifies cybersecurity tooling in critical infrastructure as high-risk under Annex III, triggering obligations for human oversight, transparency and demonstrable robustness [12]. Both DORA and NIS2 require traceable human accountability for incident classification, which stands in structural tension with the fully autonomous postures that dominate the corpus of our systematic review.

For the financial sector specifically, the review treats applicability as an evidence gap rather than a synthesis: latency expectations in fraud and trading contexts are measured in milliseconds against documented agent response times of seconds to minutes, explainability must satisfy legal rather than

technical standards, and integration requires financial schema knowledge (SWIFT, ISO 20022, FIX) absent from general-purpose training corpora.

2.5 Positioning of the CyberAid Design Against the Review

The review closes with open research directions. Several are design problems rather than research questions, and this deliverable addresses them directly (see Table 2).

Table 2: Review directions addressed by this deliverable

Open research direction	Response in this deliverable
Standardised orchestration protocols: message schemas, conflict-resolution policies, privilege tiers, audit trail schemas	Finding schema as a single output contract (Section 10.1); explicit conflict-resolution rule (Section 9.3); Security Context privilege model (Section 6.1); append-only audit schema (Section 4.5)
Incident response and forensics coverage gap	No dedicated incident-response or forensics agent in this revision (Section 8.2); response is confined to the tiered human-in-the-loop model (Section 9.4), the audit trail (Section 4.5), and the Automated Log Analysis and Event Correlation agent's evidence-preserving output. Closing the gap fully is tracked as a follow-up action (Section 13).
Compliance-aware agent design	Risk and Compliance Assessment agent (Section 8.2); regulatory mappings carried on every finding (Section 10.1); design-to-obligation mapping (Section 11.2).
Formal human-in-the-loop frameworks: decision taxonomy, handoff interface, override mechanism, audit trail	Tiered review model supplying all four components (Section 9.4)
Financial-sector agent design under DORA and EU AI Act constraints	The deliverable as a whole; latency targets traced to requirements (Section 11.1)

Two design choices follow from the review's evidence rather than from preference.

The orchestration layer adopts the centralised/hierarchical pattern as its primary structure, because the review identifies auditability as that pattern's structural advantage and auditability is the property a DORA-regulated deployment cannot trade away. Its documented weakness (i.e., orchestrator bottleneck) is mitigated by the direct-invocation path specified in Section 3.3 rather than by abandoning the pattern.

The human-in-the-loop model is treated as a first-class design element rather than an operational afterthought. Given that no study in the corpus is human-overseen and only one uses human approval, a design intended for regulated financial deployment cannot inherit the field's default posture. Section 9.4 specifies the tiered model that results.

The design also builds on an architecture paper produced within the project. The organisation-scoped runtime architecture [2] introduces the typed Security Context primitive that Section 6 specifies, established as the contract enforced at every runtime boundary. The project's position paper [13] sets out the wider platform framing within which this orchestration layer sits.

3. Architectural Positioning of the LLM Orchestration Layer

Deliverable D2.2 defines the CyberAid reference architecture: the platform boundary, the functional layers, the component landscape and the relationships between them. This section locates the LLM orchestration layer within that architecture and states which parts of it this deliverable specifies (Table 3).

3.1 Position within the CyberAid Reference Architecture

D2.2 organises the platform into six internal functional layers above the monitored financial infrastructure. The layers denote logical areas of responsibility rather than deployment tiers, so components in one layer may be deployed in different environments while retaining the same architectural role.

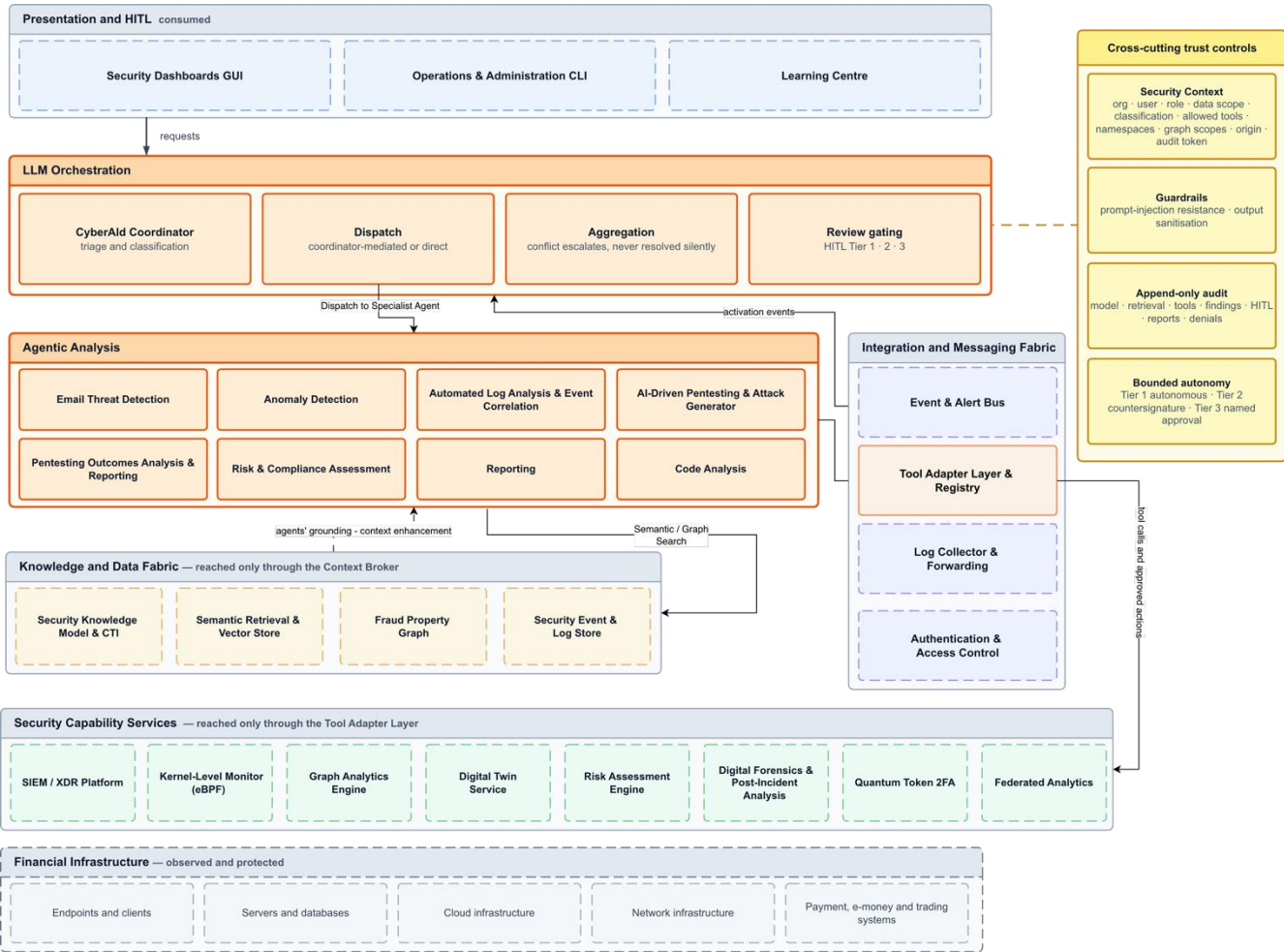


Figure 4: CyberAid reference architecture and the LLM/Agentic layers.

This deliverable spans two of those layers illustrated in coloured boxes at Figure 4. The LLM Orchestration layer receives requests from the presentation layer and activation events from the integration fabric, dispatches work to agents, aggregates and corroborates results, and applies human checkpoints before results are returned or actions proposed. The Agentic Analysis layer holds the agents themselves, which are the components performing LLM-based reasoning, tool use and human-gated analysis. Section 9 specifies the coordination behaviour of the first; Sections 4 to 8 specify the architecture, state model, governance and roles of the second.

Table 3: Relationship between the D2.2 architecture layers and this deliverable

D2.2 layer	Relationship to this deliverable
Presentation and HITL	Consumed. Dashboards and the administration CLI reach the orchestration layer via API; the review interfaces they present are specified in Section 9.4
LLM Orchestration	Specified — coordination, dispatch, aggregation and review gating (Section 9)

D2.2 layer	Relationship to this deliverable
Agentic Analysis	Specified — canonical agent architecture, state, governance and agent roles (Sections 4 to 8)
Integration and Messaging Fabric	Consumed. Event and Alert Bus, Tool Adapter Layer, Authentication and Access Control; Sections 4.3 and 3.2 specify how the orchestration layer uses them
Knowledge and Data Fabric	Consumed. Security Knowledge Model and CTI Platform, Semantic Retrieval and Vector Store, Fraud Property Graph, Security Event and Log Store; reached through the Context Broker. Section 5.5 specifies scoped access
Security Capability Services	Consumed. Reached only through the Tool Adapter Layer; never called directly by an agent (Section 4.3)
Financial Infrastructure	Out of scope/Agnostic. Observed and protected, not part of the platform

Two distinct access paths run from the agent layer downward: the Tool Adapter Layer to the Security Capability Services, and the Context Broker to the Knowledge and Data Fabric. No agent reaches either layer by any other route.

3.2 The Alerting Layer as the SIEM/Tooling Bridge

D2.2 places the Event and Alert Bus in the Integration and Messaging Fabric as the messaging backbone transporting alerts and events between detectors and orchestration consumers. The bus is developed under T4.5, which builds the alert generation and notification pipelines on a dual-channel ingestion model: high-priority alerts through a dedicated Kafka connector over the SIEM's REST API, and event data forwarded for contextual processing, with Kafka providing reliable message distribution.

The bus is what makes the platform reactive rather than interrogative. Without it, LLM analysis is something an analyst requests after noticing a problem; with it, an alert raised by the SIEM/XDR Platform, the Kernel-Level Monitor (eBPF) and Connection Tracer, a scanner or any other detector becomes a workflow trigger. Detection in CyberAId is hybrid. Pre-defined rules give fast, deterministic coverage of known patterns, while AI-based complex event processing on the Event and Alert Bus flags novel event sequences and derives new detection rules from them (D2.2). Either way, what detection produces is an alert. What has been missing is the reasoning between an alert firing and an analyst attending to it, and that is what the orchestration layer adds.

The division of responsibility at this boundary is precise. T4.5 generates, prioritises and distributes alerts. On receipt, the orchestration layer assesses severity, categorises the incident, estimates potential impact and activates the appropriate specialist agents. This triage behaviour is specified in Section 9.2. Neither side needs knowledge of the other's internals: a detector publishes to a topic, and the orchestration layer subscribes.

Events arriving on the bus are converted into workflow triggers carrying a Security Context, so an alert-initiated workflow is authorised exactly as an analyst-initiated one is (Section 3.3).

The return path does not run over the bus. Where an analysis results in an approved action, that action reaches the security services through the Tool Adapter Layer, under the enforcement and audit described in Section 4.3, rather than being published as an event. The bus carries observations upward; actions travel downward through governed adapters. This keeps the loop closed — detection to reasoning to response — with a single accountable path for anything that changes the state of a monitored system.

3.3 Orchestration Entry Points

D2.2 states that the orchestration layer receives requests from the presentation layer and activation events from the integration fabric. This deliverable specifies several entry classes covering analyst chat, administration CLI, programmatic API, alert from the bus and scheduled run.

Work reaches an agent by one of two paths. In coordinator-mediated dispatch, the CyberAId Coordinator classifies the request, selects one or more specialists, reconciles what they return and applies the review tier. This is the default and the path used whenever the work may span domains. In direct specialist invocation, an alert whose classification is unambiguous reaches its specialist without coordination. For example, an email-borne threat needs the email triage pipeline, and interposing a classification step adds latency without adding information. Section 11.1 records the requirement this serves.

The two paths differ in routing only. A directly invoked specialist performs the same enforcement, uses the same workflow state, emits the same validated finding and passes through the same review gate as one reached through the coordinator (Section 4.2). This is what makes the shorter path safe to offer. This is an optimisation of how work is routed, not a reduced-governance mode of operation.

4. The Canonical LLM Agent Architecture

The orchestration layer hosts a coordinator and an extensible set of specialist agents covering different security domains, but it contains only one execution pattern (see Figure 5). Every agent is assembled from the same runtime components and satisfies the same contract, differing in its instructions, its permitted tools, and the knowledge partition it reads, not in how it is built, secured, or observed.

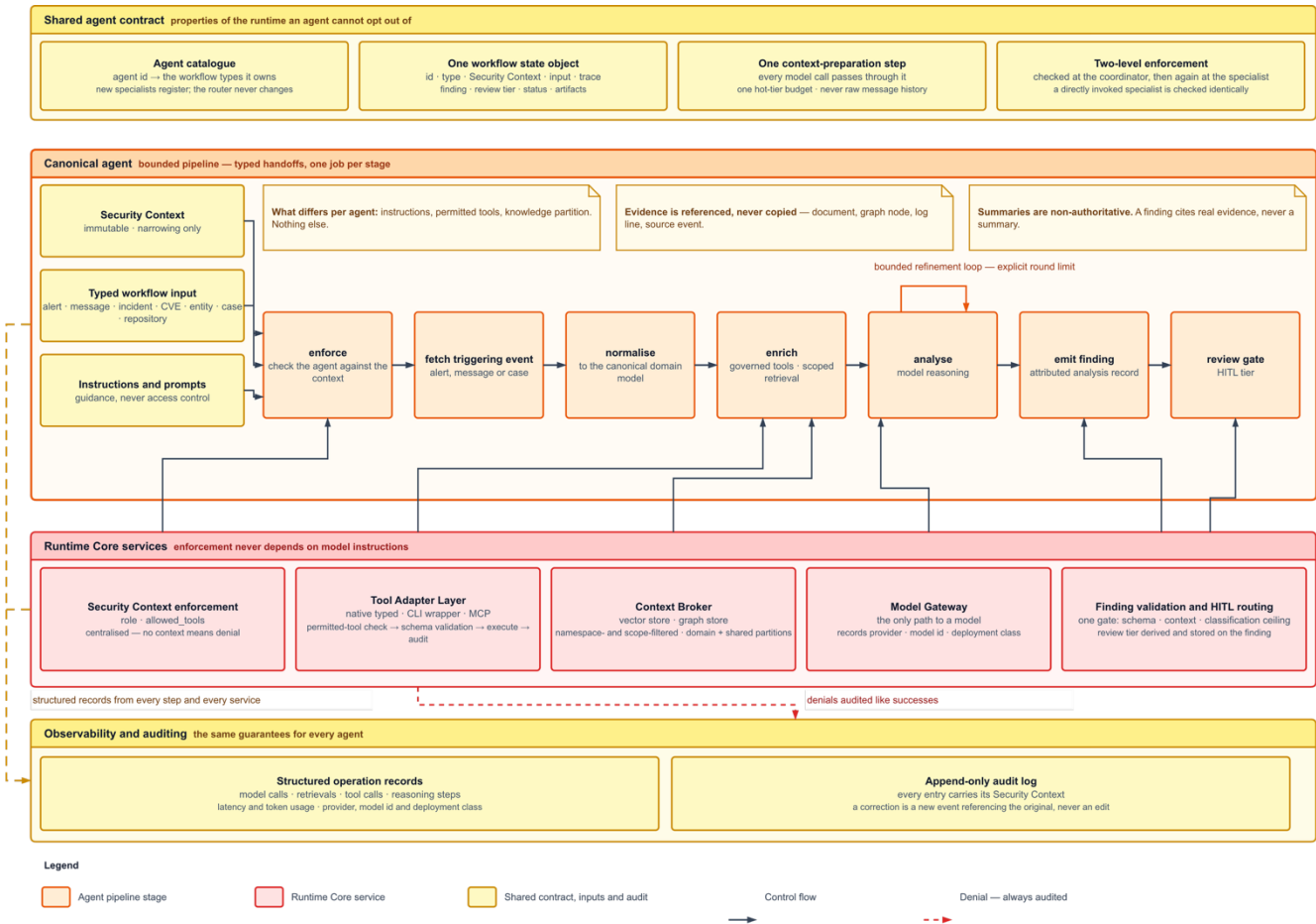


Figure 5: The canonical LLM agent with one execution pattern, shared by the coordinator and every specialist.

This is a deliberate constraint rather than an implementation convenience. If each agent implemented its own access checks, argument validation and logging, the guarantees the platform offers a regulated deployment would be only as strong as the weakest agent, and every new agent added under WP3 or WP5 would widen the review surface. Under the canonical architecture, enforcement, evidence handling and audit are properties of the runtime that an agent cannot opt out of. Adding a specialist agent adds domain capability without adding a new trust boundary.

4.1 Runtime Core vs. Orchestration Engine

The Runtime Core is CyberAid-owned code. It holds everything that must hold regardless of how a workflow happens to execute: the Security Context creation and enforcement, the Model Gateway, the Context Broker, the Tool Adapter Layer, finding validation, human-in-the-loop routing, and the audit log. The orchestration engine sits underneath it and does only graph execution and state persistence i.e. which step runs next, how intermediate state is checkpointed, and how a paused workflow resumes.

Table 4: Separation of concerns between the Runtime Core and the orchestration engine

Concern	Owner
Workflow entry from chat, CLI, API, alert or schedule	Runtime Core
Security Context enforcement at all eight boundaries	Runtime Core
Model access and provider selection	Runtime Core (Model Gateway)
Tool argument schema validation and execution	Runtime Core (Tool Adapter Layer)
Retrieval and graph access under scope	Runtime Core (Context Broker)
Finding validation and HITL tier routing	Runtime Core
Append-only audit	Runtime Core
Step sequencing and branching	Orchestration engine
Workflow state checkpointing and resumption	Orchestration engine

Two rules make the separation hold in practice. First, agents obtain model handles from the Model Gateway and reach external systems only through the Tool Adapter Layer; no agent calls a vendor SDK or an external service directly. Changing the model provider therefore becomes a configuration change confined to the gateway. Likewise, because enforcement sits in the Runtime Core, changing the orchestration engine leaves it untouched. Second, enforcement never depends on model instructions. A system prompt telling an agent not to read data outside its scope is guidance, not access control. Such checks run in the Runtime Core, before the agent’s request reaches the store.

The architecture is also designed to grow by extension rather than modification. Connector, subagent, skill and context packs add capability without changes to the Runtime Core, and every pack remains subject to the same Security Context and audit obligations as built-in components (Section 7.4).

4.2 Shared Agent Contract

Every agent is a self-contained graph whose entry step performs enforcement against the Security Context accompanying the request. Where the coordinator dispatches to a specialist, enforcement runs twice: once at the coordinator, once at the specialist. This matters for the direct-invocation path described in Section 3.3 where a specialist triggered straight from the alerting layer is checked exactly as it would be if the coordinator had dispatched it, so the shorter path is not a weaker one.

All agents share a single workflow state object, which carries the workflow identifier, the workflow type, the live Security Context, the typed inputs, the accumulated model trace, any emitted finding identifier, the human-in-the-loop requirement derived from that finding, and the workflow status. The Security Context within that state is read by every step and mutated by none; downstream components may receive a narrower view of it, never a broader one.

Agents are declared in a catalogue that maps each canonical agent identifier to the workflow types it owns. The catalogue is the single source of truth for what exists and what is routable, which keeps the dispatch logic in Section 9.2 free of hard-coded agent knowledge and lets a new specialist become reachable through registration rather than through changes to the router.

Context management is likewise shared. All agent model calls pass through one context-preparation step that enforces the token budget described in Section 5.2; no agent passes raw accumulated message history to a model. Compaction events are audited, and compacted summaries are explicitly non-authoritative. Thus, a finding must still cite real evidence and retrieval references, never a summary of them.

4.3 Tool Integration

Agents act on the world exclusively through the Tool Adapter Layer, which supports three adapter types.

Native typed adapters are the preferred form for platform-internal capability, giving a declared argument schema and a typed result the runtime can validate. Command-line wrappers cover established security tooling that exposes no usable API. MCP adapters are used for external tools and services, where a standard interface removes the need for a bespoke integration per vendor. MCP is an integration option at the platform boundary, not the internal calling convention between CyberAId agents.

Whichever type is used, every invocation follows the same sequence. Specifically, the requested tool is checked against the permitted tool set in the Security Context, arguments are validated against the adapter's schema, the adapter executes, and the call is recorded in the audit log. Rejections are audited as thoroughly as successes, since an agent repeatedly attempting calls outside its permitted set is itself a signal worth retaining.

Some adapters require a second, independent authorisation. Path-scoped filesystem tools are gated both by the permitted tool set and by explicit operator grants over specific directory roots, so an agent authorised to read files still cannot reach a path an operator has not opened. Pack-declared tools inherit the same gating; enabling a pack never widens what a Security Context permits.

4.4 Knowledge Access

Agents reach shared knowledge through the Context Broker, which allows access to the Knowledge and Data Fabric for document retrieval and structured traversal behind stable interfaces. The broker applies enforcement before delegating to either store. Retrieval is confined to the namespaces named in the Security Context and to results at or below the permitted classification, and graph traversal is confined to the permitted graph scopes.

Knowledge is partitioned rather than pooled. Each agent reads a domain knowledge partition holding the threat intelligence, regulatory texts, playbooks and other knowledge relevant to its role, alongside a shared partition holding organisation-wide cross-domain knowledge, with both filtered by the requesting context. Two analysts in the same organisation issuing an identical query therefore receive results bounded by the namespaces, data scopes and classification ceiling their respective Security Contexts carry, so a query cannot surface material the requester's role is not entitled to see. Section 5.5 covers the partitioning model and the store technology options.

Retrieved material enters findings as references (e.g., a document and section, a graph node, a log file and line range, a source event) rather than as copied content. Keeping regulated data referenced rather than duplicated bounds where sensitive material comes to rest, and it gives a reviewer a path back to the original source instead of a paraphrase of it.

4.5 Observability and Auditing

The runtime provides the same observability guarantees to every agent, which is what allows an analysis to be reconstructed after the fact by someone who did not run it.

Each agent emits structured operation records for its model calls, retrievals, tool invocations and reasoning steps, together with latency and token-usage metrics. Model invocations additionally record the provider, the model identifier and the deployment class (i.e., cloud, private cloud, local or test) and the same identifiers are attached to any finding the invocation contributed to, so a finding can be traced to the model that produced it.

Beneath this sits the audit log, which records the categories in Table 5. Every entry carries a reference to the Security Context under which it occurred, which is what links a scattered set of events into a single reconstructable workflow.

Table 5: Audit event categories recorded by the runtime

Category	Recorded events
Model	Prompts, responses, model references, latency, token usage
Retrieval	Document retrieval queries, graph queries
Tools	Tool calls, including schema rejections
Operator grants	Filesystem grant and revocation events
Findings	Finding emission and validation failures
Human-in-the-loop	Tier routing, approvals, rejections and recorded rationale
Reports	Report generation and scope checks
Denials	Every enforcement denial, with the violated invariant
Packs	Pack install, verify, enable, disable and upgrade

The log is append-only. A correction is a new event referencing the original, never an edit of it, so the record of what the system did at the time survives any later revision of what it should have done. Denial events matter as much as successful ones: an attempt to retrieve outside scope, a tool call with malformed arguments, or a finding that failed validation each leave a permanent record naming the invariant that was violated.

These properties make the design answerable to the incident documentation obligations of DORA and to the human-oversight expectations of the EU AI Act, discussed in Section 11.2. The systematic review noted in Section 2.3, identifies auditability as the principal structural advantage of centralised orchestration. With the proposed design, auditability is made explicit in the runtime rather than left to the orchestration pattern to imply.

5. State Handling, Memory and Shared Data Infrastructure

Section 4 established that agents share one execution pattern. That pattern only works if they also share one way of holding state, one definition of what an organisation’s context is, and one governed path to stored knowledge. This section specifies those three and closes with the storage technology options still open at design time.

5.1 Workflow State and Checkpointing

Every workflow, however it was triggered, carries a single state object from its first step to its last. One schema serves the coordinator and all specialists, so a coordinator can hand state to a nested specialist without translating between formats. In this way, a specialist invoked directly from the alerting layer starts from the same shape of state it would have received through the coordinator.

Table 6: Workflow state fields

Field	Purpose
workflow_id	Unique workflow identifier; also the checkpoint thread key
workflow_type	Dispatch key selecting the specialist

Field	Purpose
security_context	The live Security Context, read by every step and mutated by none
input	Typed workflow inputs (alert, message, case, entity, repository, ...)
messages	Accumulated model interaction trace
finding_id	Identifier of the finding emitted, where the workflow produces one
hitl_requirement	Review tier derived from the finding
status	running, completed, pending_hitl or failed
artifacts	Pointers to tool results and retrieval references, not the payloads themselves
step_brief	Supplementary summary written by context compaction (Section 5.2)

Two properties of this schema matter for a regulated deployment. The artifacts field holds references rather than content, so bulk tool output and retrieved documents do not accumulate inside workflow state. Instead, they stay in their systems of record and are cited. And the Security Context is carried in state but never modified there; a workflow cannot escalate its own permissions mid-execution, because the object granting them is fixed at entry.

State is checkpointed after each step, keyed by the workflow identifier. As a result, a workflow paused at a review gate can resume later on a different process, since its state is durable rather than held in memory. In addition, a workflow that fails part-way can be inspected at the step where it stopped rather than only through its logs.

Checkpoint state is kept in a separate store from the audit log and findings. The two have different obligations: checkpoints are working state, subject to normal lifecycle management, while audit records are append-only and retained (Section 4.5). Mixing them would make the retention policy of the audit log hostage to the storage behaviour of transient execution state.

5.2 Context Policy: Hot, Warm and Cold Tiers

Long-running security analyses generate more material than any model context window can hold. Rather than letting agents decide individually what to keep, the runtime applies one policy across three tiers.

Table 7: Context tiers

Tier	Contents	Bounded by
Hot	Messages passed to the model for the current step, plus the step brief	Explicit message budget
Warm	Workflow state in the checkpoint: identifiers, references, status	Workflow lifetime
Cold	Audit records, findings, and the knowledge held in the Knowledge and Data Fabric	Never compacted away

Every agent model call passes through a single context-preparation step that enforces the hot-tier budget. This is the only place the budget is applied, which is what stops the policy fragmenting into per-agent variations as new specialists are added under WP3 and WP5.

When a workflow exceeds its hot-tier budget, older turns are evicted under one of two strategies. Truncation drops them and records how many were omitted. Summarisation sends the evicted turns to the model with instructions to preserve identifiers, decisions and evidence references without introducing new facts, and stores the result as the step brief. Either strategy emits a compaction audit event recording the workflow, the strategy applied, and the number of turns evicted, so a later reviewer can see that material left the working context and how.

The step brief is explicitly non-authoritative. A finding must ground its claims in evidence and retrieval references pointing at real sources, never in a summary of earlier reasoning. This distinction is what keeps

compaction a performance measure rather than a quiet weakening of the evidence chain. The idea is that a summary can guide the next step, but it cannot become the basis of a reported conclusion.

5.3 Organisation Context via CYBERAID.md

A deployment needs to know things about the organisation it protects that are not access decisions: which regulatory regimes apply, which systems are critical, who is contacted at which severity, what internal terminology means, how material is classified, and which playbooks apply to which situation.

This is held in a single reviewable file, CYBERAID.md, loaded into agent prompts and retrieval context. Keeping it in one readable document rather than distributing it through prompts has a practical benefit: a compliance officer can read what the system believes about their organisation, and can correct it, without reading code or prompt templates.

CYBERAID.md supplies context and never grants permissions. The Security Context supplies authority. Text in the profile stating that an agent may access a system has no effect on whether the runtime permits that access. Without this rule the file would become an unreviewed second authorisation path, editable by anyone who can edit a document.

The design intent is that onboarding a new organisation requires configuration and an organisation profile, with no change to platform code.

5.4 Scoped and Longer-Term Memory

Workflow state ends with its workflow. However, some knowledge should outlive it. For example, a behavioural baseline for a counterparty, a correction an analyst made to an earlier finding, the normal shape of traffic from a given system. Without somewhere to put this, every analysis restarts from zero and the platform cannot improve from the work analysts have already done.

Memory is scoped, not global. Reads and writes are checked against the organisation and data scope in the Security Context, which is one of the enforcement points listed in Section 6.1. Within a deployment, a behavioural baseline written under one data scope is not readable from another, and this is enforced at the boundary rather than by convention in the memory implementation. The `org_id` check on every memory read and write binds each record to the organisation on whose authority it was created.

Retention is bounded and purging is audited. The design position is that stored observations expire on a configured horizon by default rather than persisting indefinitely, and that expiry runs as a recorded event rather than a silent deletion. Where memory holds observations relating to identifiable individuals, this bounded retention is what supports erasure obligations under the GDPR (Section 11.2).

Analyst corrections are the memory writes with the clearest operational value and the highest sensitivity: they encode judgement, and a poisoned or mistaken correction propagates into later analyses. The design treats correction writes as scoped, attributed and audited, so a later reviewer can establish which prior judgement influenced a given finding and who made it.

5.5 Shared Knowledge Infrastructure and Common Tooling

All agent access to stored knowledge (Knowledge and Data Fabric Layer of Figure 4) runs through the Context Broker, which exposes a vector store for document retrieval and a graph store for structured traversal over entities and relationships. Agents depend on these interfaces rather than on a particular database, which is what lets the storage technology change without touching agent logic.

The scoping model in this design is internal to a single organisation. A CyberAid deployment serves one financial organisation, and the knowledge, graphs and memory it holds are that organisation's own. Namespaces and graph scopes therefore separate domains, sensitivity levels and business units inside the organisation rather than separating one organisation's data from another's. The `org_id` field of the Security Context identifies the deploying organisation and is carried on every context, retrieval and memory record, so that attribution and audit remain unambiguous and the same enforcement path applies unchanged

should a deployment later serve more than one legal entity, for example a banking group operating several separately regulated subsidiaries.

Retrieval is namespace-scoped and traversal is graph-scope-scoped, with both checked against the Security Context before the query reaches the store. Each agent reads a domain partition holding the knowledge relevant to its role (see Section 8.2 for the current roster) alongside a shared partition of cross-domain knowledge. Partitioning keeps retrieval relevant as well as safe. For instance, an agent querying a partition sized to its domain returns better results than the same query against an undifferentiated store.

Knowledge returned by the Context Broker follows the retrieval-result representation specified in D2.4. The Security Context's retrieval namespaces are evaluated against an element's knowledge partition and semantic domain, its graph scopes against the element's graph scope, and its classification ceiling against both the element's classification and its handling constraints, including TLP markings.

Three storage decisions remain open at design time. They are recorded at Table 8 with the options under consideration and what should decide between them. The interface boundary is what makes deferring these decisions safe. Because agents reach storage only through the broker's interfaces, each choice changes an implementation behind a stable contract rather than the design specified in this deliverable.

Table 8: Open technology options for the shared data infrastructure

Decision	Options under consideration	Deciding factor
Relational store	Embedded SQLite; PostgreSQL	Pilot-scale single-node deployment versus organisation-wide operation, concurrent write volume, and the organisation's own database operations standards
Vector store	Embedded SQLite vectors; Qdrant; Chroma; pgvector	Volume of indexed knowledge, filtered-query performance, and whether the deployment can host an additional service
Graph store	SQL-backed graph over the relational store; Neo4j	Traversal depth and query complexity required by threat-intelligence and log-correlation workflows (the Fraud Property Graph is specified separately on Neo4j in D2.2)

LightRAG, adopted by the Automated Log Analysis and Event Correlation agent (D2.2), operates over whichever vector and graph backends are selected.

6. Security Context and Governance Model

The Security Context is the contract that makes the rest of this design enforceable. As discussed at Sections 4, agents are gated by it, retrieval and traversal are filtered by it, memory is partitioned by it, and audit records are correlated through it. This section specifies what it contains, where it is checked, and what it does not cover. Figure 6 shows the schema, enforcement points, and audit flow.

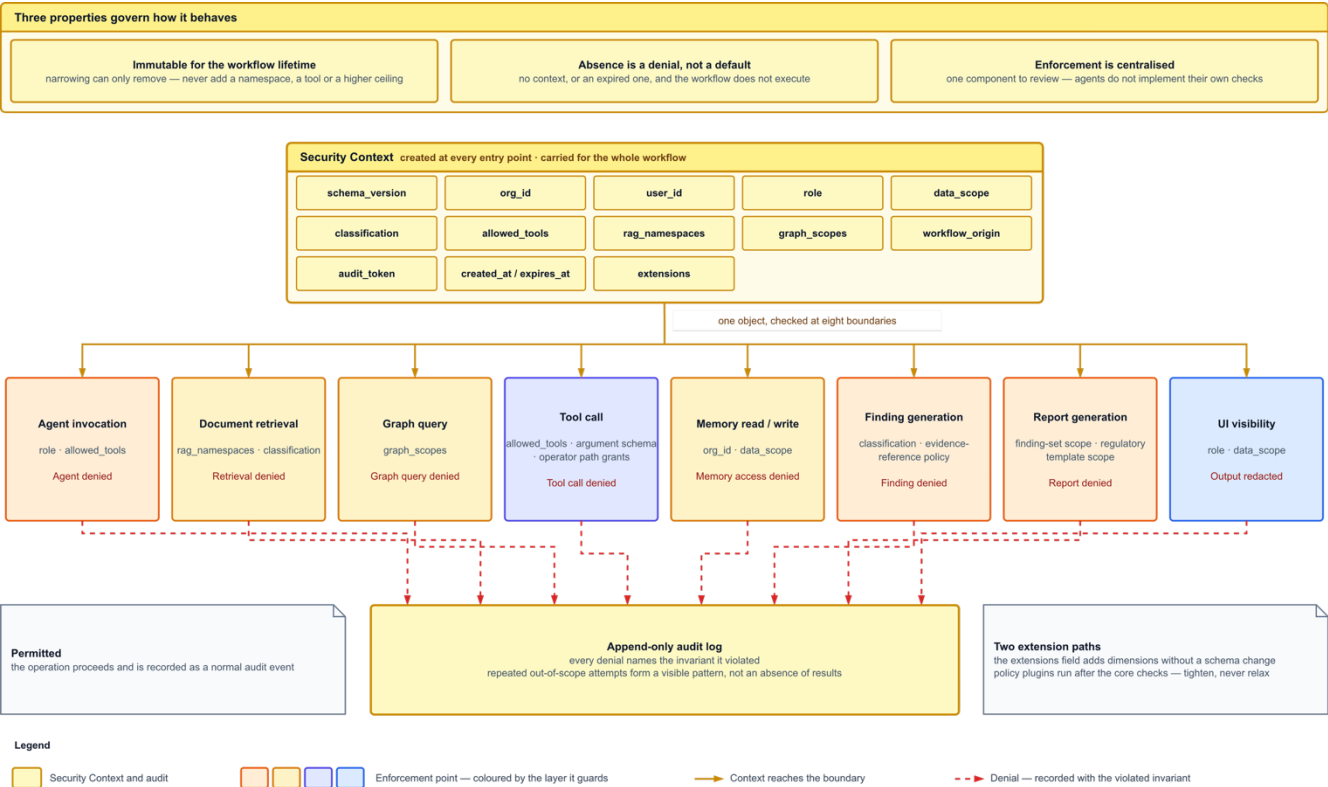


Figure 6: Security Context schema, enforcement points, and audit flow

6.1 Security Context Schema and Enforcement Points

A Security Context is created at every entry point (i.e., analyst chat, command line, API call, alert from the monitoring layer, scheduled run) and accompanies the workflow it starts for that workflow’s entire life. Table 9 lists the fields it carries.

Table 9: Security Context fields

Field	Purpose
schema_version	Contract version, so a context can be validated against the schema it was issued under
org_id	The organisation on whose authority the workflow acts
user_id	The acting user
role	Role used for authorisation decisions
data_scope	Data scopes the workflow may reach
classification	Maximum classification permitted in outputs, including handling constraints such as TLP (D2.4)
allowed_tools	Tool adapters the workflow may invoke
rag_namespaces	Document namespaces available to retrieval
graph_scopes	Graph scopes available to traversal
workflow_origin	Entry class: chat, CLI, API, alert or schedule
audit_token	Correlates every audit event belonging to this workflow
created_at / expires_at	Issue and expiry times
extensions	Forward-compatible optional dimensions

Three properties govern how the Security Context of the LLM Orchestration and Agents behaves:

It is immutable for the lifetime of the workflow. No step can rewrite the object that authorises it. Where a component needs a reduced view — a specialist that should not inherit the coordinator’s full tool set, for instance — it receives a narrowed context. Narrowing can only remove; it cannot add a namespace, a tool, or a higher classification ceiling. Delegation in this design therefore travels in one direction, and the permissions of any nested step can be reasoned about from the context that entered at the top.

Its absence is a denial, not a default. A boundary reached without a context, or with an expired one, denies rather than falling back to a permissive baseline. This matters most on the paths that are easiest to overlook: a scheduled run at 03:00 and an alert arriving from the monitoring layer both carry a context or do not execute, exactly as an interactive analyst request does.

Enforcement is centralised. The checks live in the Runtime Core, and agents do not implement their own. An agent cannot weaken a check it does not perform, and a reviewer establishing what the platform enforces reads one component rather than one per agent. Table 10 lists the enforcement points and the invariant checked at each.

Table 10: Enforcement points and the invariants checked at each

Enforcement point	Invariant checked	Denial
Agent invocation	role, allowed_tools	Agent denied
Document retrieval	rag_namespaces, classification	Retrieval denied
Graph query	graph_scopes	Graph query denied
Tool call	allowed_tools, argument schema, operator path grants	Tool call denied
Memory read/write	org_id, data_scope	Memory access denied
Finding generation	classification, evidence-reference policy	Finding denied
Report generation	Finding-set scope, regulatory template scope	Report denied
UI visibility	role, data_scope	Output redacted

Every denial names the invariant it violated and is written to the audit log (Section 4.5). A denial is a recorded security event, not a silent failure. As a result, an agent making repeated attempts outside its scope produces a visible pattern rather than an absence of results.

Two extension paths are built into the schema. The extensions field carries additional dimensions without a schema version change, and a policy plugin interface allows deployment-specific checks to run after the core checks. These checks always run after the core checks, never instead of them, so a plugin can tighten enforcement but cannot relax it. A deployment serving more than one legal entity — a banking group whose subsidiaries are separately regulated, for instance — is the case this anticipates. It would require an explicit entity dimension alongside org_id, which is a scoping addition rather than a redesign.

6.2 Guardrails Framework

The Security Context answers one question well: is this workflow permitted to reach this resource? It does not answer a second one: should this output be trusted and acted upon? Authorised access can still produce a wrong answer, and an attacker who cannot cross a scope boundary may still influence content that an authorised agent legitimately reads.

Separating these two concerns is deliberate, because they need different mechanisms and different evidence. Table 11 summarises this division of responsibility between enforcement and guardrails.

Table 11: Division of responsibility between enforcement and guardrails

Concern	Handled by
Retrieval outside permitted namespaces or organisation	Security Context enforcement

Concern	Handled by
Tool calls outside the permitted set, or with invalid arguments	Security Context enforcement
Memory writes outside the permitted organisation and data scope	Security Context enforcement
Unaudited execution at any boundary	Security Context enforcement
Prompt injection carried inside authorised content	Guardrails
Plausible but unsupported analysis	Guardrails, evidence grounding, confidence thresholds
Valid tool calls driven by attacker-influenced input	Guardrails, human-in-the-loop tiers
Sensitive content reaching an output it should not	Guardrails, classification ceiling

Guardrails operate at agent boundaries in both directions. On the way in, content entering an agent's context is validated and screened for injected instructions. This threat is unavoidable in this domain rather than incidental to it. For example, a phishing email under triage, a log file under correlation analysis, and a threat intelligence report under enrichment are all attacker-authored or attacker-influenced by nature. An agent reading them is doing its job. The design assumption is therefore that retrieved content is data to be analysed and never instruction to be followed, and that the boundary between the two is enforced by the runtime rather than by asking the model in a system prompt.

On the way out, four checks apply before an agent's output becomes a finding. Output sanitisation removes content that should not leave the boundary. Grounding requires that claims in a finding trace to evidence and retrieval references, which is the practical defence against hallucinations (an assertion with no reference behind it does not become a reported conclusion). The classification ceiling in the Security Context bounds what may appear in output at all. And calibrated confidence, carried as a field on every finding, feeds the tier routing in Section 9.4, so low-confidence work reaches a human rather than an action.

These mechanisms reinforce each other rather than standing alone. Confidence scores drive review tiers; evidence references make review possible in reasonable time; the audit trail records what was checked and what was decided. A finding that survives all four has a reviewable justification attached to it, which is the property a regulated deployment needs and a bare LLM response cannot offer.

Having said that, injection detection is a mitigation and not a guarantee. The systematic review discussed in Section 2.4 identifies prompt injection and adversarial robustness as unresolved across the field, with no published defence achieving reliable protection. Additionally, confidence calibration in security contexts remains an open research problem. A well-formed confidence score is not the same as a well-calibrated one. The design responds to both by not relying on either. The tiered human-in-the-loop model in Section 9.4 exists precisely because guardrails are imperfect, and high-impact actions are routed to human approval regardless of how confident an agent reports itself to be.

7. PROPOSED TECHNOLOGY STACK FOR THE ORCHESTRATION LAYER

The preceding sections specify behaviour without naming products, which is what keeps the design reviewable independently of any vendor. This section names the technologies proposed to implement that behaviour, so T5.1 and the other agent-building tasks across WP3-5 start from a concrete baseline rather than re-deriving one.

Four constraints shaped the selections. The platform must be deployable inside a financial institution's own infrastructure, since several pilot participants cannot send security telemetry to an external service. It must be model-agnostic, because the model a bank will approve is a procurement outcome rather than an engineering one. Every component must be able to report what it did, to satisfy the audit obligations in Section 4.5. And the runtime must not be locked to one vendor's abstractions, since the deliverable's design lifetime exceeds the release cycle of every product named below and summarised at Table 12.

Table 12: Proposed technologies by layer

Layer	Proposal	Reason
Orchestration engine	LangGraph	Nested subgraph composition, per-step checkpointing, and interrupt support for review gates
Backend service	FastAPI (async) with Pydantic	Async-native, typed request and response contracts, machine-readable API generation
Relational persistence	SQLAlchemy 2.0 (async) with Alembic	Repository pattern behind a stable interface; versioned schema migration
Model access	Provider registry behind a gateway interface	Vendor change confined to configuration (Section 7.2)
Knowledge stores	Vector and graph interfaces with pluggable backends	Storage choice deferred behind a contract (Section 5.5)
Command-line interface	Typer	Same typing conventions as the service layer
Operator console	React with TypeScript, Vite, Tailwind CSS	Strict typing across the UI boundary; streaming interaction for analyst chat
API contract	OpenAPI 3.1	Machine-readable, validated in continuous integration

Proposals for the orchestration layer. Components owned by other tasks — notably the Security Knowledge Model and CTI Platform (D2.4) — are not constrained by this table.

7.1 Orchestration Engine and Backend Framework

The orchestration engine supports the composing of specialist graphs inside a coordinator graph without translating state between them and checkpointing state at each step so a workflow can be paused and resumed, and interrupting execution before an action to await human approval. LangGraph provides all three, which is why it is proposed here.

The Runtime Core boundary described in Section 4.1 is what keeps this a replaceable choice. LangGraph handles execution and state; it does not perform enforcement, hold the audit log, validate findings, or reach external systems. A different engine would change how steps are sequenced, not what the platform guarantees.

The service layer is proposed as FastAPI with Pydantic models, and persistence as SQLAlchemy 2.0 in async mode with Alembic for migrations. Typed models at the API boundary matter more here than in a typical service. The same objects carry the Security Context and the finding schema, so a malformed context or an incomplete finding fails at the boundary rather than partway through an agent’s reasoning.

7.2 Model Gateway and Provider Registry

Every model call passes through the Model Gateway. Agents request a model handle for a stated purpose, such as triage, reporting, enrichment, or summarisation, and receive one; they never construct a provider client. Changing the model or the vendor is a configuration change confined to the gateway. The reason is procurement, not portability. For example, a bank’s approval of a model backbone depends on its data-processing agreements, its supervisory expectations, and where inference physically runs, considerations that vary by institution and change over time. A design that binds agent logic to one provider forces re-engineering each time that answer changes.

Table 13: Model provider classes supported by the gateway

Deployment class	Providers	Relevance for a financial deployment
Cloud, direct	OpenAI, Anthropic, Google	Fastest capability access; requires an accepted data-processing agreement
Private cloud	Azure OpenAI, AWS Bedrock, Google Vertex AI	Inference inside the institution's own tenancy and region
Local	Ollama, OpenAI-compatible endpoints such as vLLM or an internal proxy	No data leaves the premises; supports the strictest residency constraints
Test	Deterministic stub	Offline evaluation with no network dependency and no cost

To support reproducibility, every invocation records the provider, the model identifier, the deployment class, latency and token usage, and those identifiers travel onto any finding the call contributed to (Section 4.5). Secrets are supplied through environment variables or a secret manager, never through configuration files held in version control.

7.3 Persistence and Knowledge Stores

The technologies named in this section are proposals for the orchestration layer's own persistence and for the interfaces through which it reaches stored knowledge. They are not implementation requirements for the Security Knowledge Model or for the CTI Integration Framework, whose technology choices are owned by T2.3 and specified in D2.4. Where this deliverable names a store, it does so to give T5.1 and the agent-building tasks a concrete starting baseline; the binding element is the interface contract in Section 5.5, not the product behind it.

Section 5.5 leaves storage decisions open with stated criteria. These stores form part of the Knowledge and Data Fabric in CyberAld's reference architecture (D2.2). The proposal for the design phase is to start embedded and migrate under load. An embedded relational database, embedded vector storage, and a graph implemented over the relational store are enough to exercise every workflow in this design, and they let a pilot site run the platform without provisioning additional infrastructure.

Production deployment is expected to move the relational store to PostgreSQL, vector search to a dedicated vector database, and graph traversal to a graph database, once the volume of indexed knowledge and query complexity justify the operational cost. Because agents reach storage through the interfaces in Section 5.5, each migration replaces an implementation rather than changing agent code.

Workflow checkpoints and the audit log stay in separate stores regardless of backend, for the reasons given in Section 5.1.

7.4 Tool Integration Technologies

Section 4.3 defines three adapter types. Specifically, native adapters are implemented in the platform's own language with declared argument schemas validated on every call; command-line wrappers execute established security tooling with the same validation and audit path applied around them; and MCP adapters connect external tools and services that expose an MCP interface.

MCP earns its place at the external boundary specifically. The security tooling a financial institution runs is heterogeneous and largely predates any agent standard, and a shared protocol is what avoids writing a bespoke integration for each product. Inside the platform, a native adapter with a typed schema gives stricter validation and a shorter call path, so MCP is not used where an internal adapter would do.

Extension packs (or agent skills) declare the tool identifiers, retrieval namespaces and graph scopes they require in a manifest, which lets an operator see what a pack would be able to reach before enabling it. The design position is that manifests are signed and that a pack claiming scope beyond what an operator has registered is refused at install, with enable and disable recorded as audit events.

7.5 Alerting/Event Bridge and Operator Interfaces

The alerting bridge described in Section 3.2 is realised as a Kafka topic set carrying alerts from the SIEM/XDR Platform, the Kernel-Level Monitor (eBPF), and scanning tools into the orchestration layer. The Event and Alert Bus is built under T4.5 on Kafka, with a dedicated connector for high-priority alerts and event data forwarded for contextual processing.

Three operator-facing interfaces are proposed. A command-line interface covers configuration, evaluation runs and audit export, and is the interface most likely to be used by the engineers deploying the platform. A web console provides analyst chat with streaming responses, findings review, workflow triggering, and management of the operator grants described in Section 4.3. And an OpenAPI 3.1 contract defines the HTTP surface, generated from the same typed models the service uses and validated in continuous integration so the published contract cannot drift from the implementation. This is the interface the other CyberAid components integrate against.

8. AGENT SPECIFICATIONS

This section specifies each agent at the level the design phase can justify, including its domain, its scope boundary, and the task responsible for building it. The specialist agents are being built across technical WPs of CyberAid by tasks that own the underlying security capability, and their detailed behaviour will be settled during implementation and evaluation phase of the project.

What this deliverable does fix applies to every agent regardless of domain: the canonical architecture in Section 4, the shared state and context policy in Section 5, the Security Context and guardrails in Section 6, the finding contract in Section 10.1, and the review tiers in Section 9.4. Each task building a cybersecurity agent will design its internal logic, tools and reasoning approach.

Specifically, T5.1 delivers the shared runtime, the CyberAid Coordinator and the agent framework, together with reference implementations of the specialist agents; the domain capability each agent reasons over is developed in the owning WP3, WP4 or WP5 task, which delivers the domain logic and tool bindings on that shared runtime. Section 8.4 maps these roles to the tasks and partners responsible for them, and to the components D2.2 lists in the Agentic Analysis layer, summarised in Table 14 below.

Table 14: Agent roster and workflow types

Agent	Domain	Workflow types
CyberAid Coordinator	Triage, dispatch, aggregation, analyst chat; hosts the tiered human-in-the-loop runtime (Section 9.4)	Coordination across all types
Email Threat Detection	Phishing, business email compromise and malicious-content triage over email	email_triage
Anomaly Detection	User, entity and system behavioural baselining and deviation flagging	behavioral_analysis
Automated Log Analysis and Event Correlation	Log normalisation, correlation, and maintenance of incident and case relationships over the Security Event and Log Store and the security knowledge reached through the Context Broker	log_correlation
AI-Driven Pentesting & Attack Generator	Red-team scenario generation against the Digital Twin	attack_simulation
Pentesting Outcomes Analysis and Reporting	Exploitability scoring and remediation reporting over attack-simulation and code-analysis results	pentest_outcome_analysis
Risk and Compliance Assessment	Asset-context risk and exploitability scoring, and regulatory control-gap identification	vulnerability_assessment, compliance_mapping

Agent	Domain	Workflow types
Code Analysis	Static and dynamic code analysis, CI/CD review	code_analysis
Reporting	Report composition across findings	regulatory_notification, incident_narrative, posture_summary, mitigation_recommendations , analyst_query

8.1 CyberAId Coordinator (Central Router Agent)

The coordinator receives work from every entry point, classifies it, dispatches it to the appropriate specialist or specialists, reconciles what comes back, and applies the review tier before anything reaches an analyst or an action. It also hosts the analyst chat interaction, where an analyst's request is resolved into an allow-listed set of operations rather than an open instruction to a model.

Specifically, the coordinator routes and reconciles, and does not perform domain analysis itself. Domain reasoning belongs to the owning specialist even when the coordinator could plausibly attempt it, because a coordinator that quietly does specialist work becomes both a bottleneck and lacks domain expertise. The coordinator and the shared runtime beneath it are delivered by T5.1; the tiered human-in-the-loop implementation it hosts (Section 9.4) is delivered by T5.3.

8.2 Specialist Agents Overview

Email Threat Detection triages phishing, business email compromise and other malicious-content indicators arriving over email, and enriches what it finds against the Security Knowledge Model and CTI feeds specified in D2.4. It is delivered under T4.3 as the EMTD framework, an ontology-driven multi-agent pipeline specified in D2.2; its internal Router and sub-agent decomposition are nested beneath the canonical agent contract rather than replacing it.

Anomaly Detection establishes baselines for users, entities and systems, and reports departures from them. It is the agent most dependent on the scoped memory described in Section 5.4, since a baseline is accumulated rather than computed on demand. It is service-backed by the anomaly detection and privacy-preserving analytics capability delivered under T4.3.

Automated Log Analysis and Event Correlation normalises and correlates security events and logs and maintains the incident and case relationships used across the platform. Its retrieval-augmented approach over the project's security knowledge is specified in D2.2. It is realised as a native subagent on the shared runtime, delivered under T4.4.

AI-Driven Pentesting & Attack Generator produces red-team scenarios and simulated attacks against the Digital Twin Service, exercising the platform's detection and response capabilities under controlled conditions. The penetration test is designed around parameters set to describe the type of attack and steps taken into account the Digital Twin structure to create a native integration adaptable to its available information.

Pentesting Outcomes Analysis and Reporting interprets the results of attack-simulation runs, scores exploitability in the deployment's own context, and reports remediation priorities. It is service-backed by the pentest-analysis outputs of T3.6; whether it wraps that output natively or through a thinner adapter is open.

Risk and Compliance Assessment scores risk and exploitability in asset context, and identifies regulatory control gaps against DORA and NIS2. The assessment itself is performed by the Risk Assessment Engine (OLISTIC), a separate component of the Security Capability Services layer that this agent reaches as a tool through the Tool Adapter Layer; the agent supplies scope enforcement, evidence referencing, severity and confidence expressed in platform terms, and a validated finding. It is delivered under T5.2 and combines

two patterns: service-backed over the Risk Assessment Engine for vulnerability assessment, and native reasoning over regulatory texts for compliance mapping.

Reporting composes findings into documents for a stated audience: regulatory notifications, posture summaries, incident narratives. It works from the findings specialists have emitted and queries their owning specialists directly rather than through the coordinator, so that report generation does not contend with active incident processing. Its constraint is that it reports what findings support and does not introduce assessments of its own. It is delivered under T5.1.

Code Analysis examines source code and CI/CD configurations for malicious patterns, insecure practices and supply-chain exposure, producing structured findings traceable to established control frameworks. It is delivered by T3.3 as a native subagent inside the shared agentic runtime, exposed over MCP.

8.3 Agent Realisation Patterns

Building a specialist agent does not always mean building a complete reasoning system fully leveraging the proposed canonical agent. A native agent implements domain reasoning directly: it plans, calls tools, interprets results and forms a judgement using the model gateway. Email Threat Detection and Code Analysis follow this pattern. Meanwhile, some functions are better served by a thin wrapper agent over a standalone service using the existing runtime, rather than a native reasoning system.

A service-backed agent wraps a capability another CyberAid task develops. The analysis itself is performed by that service (e.g., graph analytics, anomaly detection, risk assessment, log correlation) and the agent supplies what the service does not: scope enforcement, evidence referencing, confidence and severity in platform terms, and a valid finding. This pattern matters for effort estimation. A partner whose detection capability already exists is building an adapter with proper prompt engineering, not a second detection system.

An external-tool agent wraps a CyberAid cybersecurity tool reached through the MCP, command-line or tool adapter types described in Section 4.3. The governance is identical: an external tool is still invoked under the permitted-tool set, still schema-validated, still audited.

The three patterns are not exclusive, and the more capable agents will combine them. For example, the Automated Log Analysis and Event Correlation agent may call a task-developed correlation engine and an external SIEM query tool, then apply its own reasoning over the results within a single workflow. What they share is the contract. Whatever sits behind an agent, what leaves it is a validated finding carrying its evidence, and what governs it is the Security Context.

8.4 Agent-to-Task Mapping

The mapping below records, for each agent role, the capability it reasons over, the task and partner responsible for delivering it, and the realisation pattern expected. It is grounded in the task descriptions of the Grant Agreement and reconciles the role view used in this deliverable with the component view used in D2.2.

Table 15: Agent roles mapped to implementing tasks and partners

Agent role	Backing capability	Task	Lead	Pattern
CyberAid Coordinator	Orchestration runtime, triage and dispatch; tiered HITL implementation	T5.1, T5.3	INNOV, UPRC	Native
Email Threat Detection	Email content analysis; consumes the Security Knowledge Model and CTI feeds (D2.4)	T4.3	UPRC	Native

Agent role	Backing capability	Task	Lead	Pattern
Anomaly Detection	Anomaly detection and privacy-preserving analytics	T4.3	UPRC	Service-backed
Automated Log Analysis and Event Correlation	Log ingestion, normalisation and correlation rules; retrieval-augmented correlation over the security knowledge; incident and case relationship maintenance	T4.4	UPRC	Native
AI-Driven Pentesting & Attack Generator	Red-teaming platform, reaching the Digital Twin Service as its target environment	T3.1	UNP	Native
Pentesting Outcomes Analysis and Reporting	Pentest and attack-simulation outcome interpretation, exploitability scoring	T3.6	UNP	Open — native vs. a thin wrapper over a standalone pentesting service
Risk and Compliance Assessment	Regulatory texts and the DORA and NIS2 reporting templates, read from the knowledge layer through the Context Broker	T5.2	UBI	Hybrid — service-backed (Risk Assessment Engine) and native (compliance mapping)
Code Analysis	LLM code and pipeline analysis exposed over MCP	T3.3	INNOV	Native
Reporting	Report composition across findings	T5.1	INNOV	Native
Graph Analytics Engine	Graph-based fraud pattern detection over the Fraud Property Graph	T4.1	IQB	Reached as a tool
Digital Twin Service	Simulated environment used by the pentesting/attack-generation agent	T3.2	UBI	Reached as a tool
Risk Assessment Engine (OLISTIC)	Asset-centric risk, dependency and exploitability assessment; attack-path analysis	T5.2	UBI	Reached as a tool
Digital Forensics & Post-Incident Analysis	Evidence collection and timeline reconstruction	T5.4	K3Y	Reached as a tool — a thin wrapper agent over this service is possible

Agent role	Backing capability	Task	Lead	Pattern
Alert generation	Rule-based and AI-based (CEP) alert generation; event bus feeding the orchestrator	T4.5	UNP	Upstream producer

Five rows are not agents and are listed for completeness. The Graph Analytics Engine, the Digital Twin Service and the Risk Assessment Engine are capabilities agents reach through the Tool Adapter Layer rather than agents in their own right; the Digital Forensics & Post-Incident Analysis capability is the same, with a thin wrapper agent over it possible but not committed in this revision (Sections 8.2 and 13). Alert generation sits upstream: T4.5 produces the alerts that trigger workflows (Section 3.2) rather than consuming them.

Two consequences follow for the agents named here. Each owns the domain behaviour like instructions, tool bindings, thresholds and domain evaluation. Yet, none owns enforcement, state handling, audit or the finding contract, which the shared runtime provides. And because the runtime is common, an agent added later inherits the governance and observability properties of the ones before it without re-implementing them.

9. ORCHESTRATION LOGIC, WORKFLOWS AND HUMAN-IN-THE-LOOP DESIGN

This section specifies how work reaches the right agent, what happens when several agents have something to say about the same event, and where a human enters the loop.

9.1 Orchestration Patterns in Use

Section 2.3 set out the three multi-agent patterns the systematic review identifies. This design uses two of them, at different levels, and declines the third.

At the platform level the pattern is centralised and hierarchical: the coordinator holds the picture of the work, decides which specialists to activate, and reconciles what they return. The review reports this pattern’s principal advantage as auditability — a single component records which agents were activated, on what input, with what result — and that is the property a DORA-regulated deployment needs most (Section 11.2). The known weakness is equally clear: a central coordinator is a bottleneck and a single point of failure. The direct-invocation path in Section 3.3 is the mitigation, letting a well-classified alert reach its specialist without waiting for coordination it does not need.

Inside a specialist the pattern is a bounded pipeline. A workflow moves through defined stages with typed handoffs — enforcement, retrieval of the triggering event, normalisation, enrichment, analysis, finding emission, review gate — so each stage has one job and the point of failure is identifiable when something goes wrong. Where a stage needs to iterate, as the Automated Log Analysis and Event Correlation agent does when refining a search over evidence, the loop is bounded by an explicit round limit rather than left to the model to decide when it has finished.

Debate and consensus is not adopted. The review credits it with strong reliability gains, and that is a real argument in its favour. It is declined because it multiplies inference cost and latency in a domain with detection-time expectations, convergence is not guaranteed and it complicates the audit record. Explaining to a supervisor that three model instances disagreed and a vote resolved it is a harder story than a specialist’s finding with its evidence attached. The Reporting agent’s direct access to specialists (Section 8.2) gives a comparable benefit for the case that most needs it, without the ambiguity.

Across both levels the orchestration supports automatic triage on arriving events, manual triggering by an analyst, parallel dispatch to several specialists, sequential chaining where one result feeds another, review checkpoints, bounded refinement loops, escalation when confidence is too low to recommend, and scheduled periodic analysis.

9.2 Workflow Type Catalogue and Dispatch Logic

A workflow type is the dispatch key. Each type names one specialist and one pipeline, which keeps routing a lookup rather than an inference.

Table 16: Workflow type catalogue

Workflow type	Owning agent	Typical trigger
email_triage	Email Threat Detection	Alert or reported message
behavioral_analysis	Anomaly Detection	Anomaly signal, scheduled baseline review
log_correlation	Automated Log Analysis and Event Correlation	Alerting bridge event stream, scheduled correlation run
attack_simulation	AI-Driven Pentesting & Attack Generator	Scheduled red-team run, analyst-triggered pentest request
pentest_outcome_analysis	Pentesting Outcomes Analysis and Reporting	Attack-simulation run completion, external pentest tool output, code-analysis finding
vulnerability_assessment	Risk and Compliance Assessment	Scanner output, CVE publication, periodic risk reassessment
compliance_mapping	Risk and Compliance Assessment	Analyst request, or chained from a finding
code_analysis	Code Analysis	Pipeline event, repository scan request
regulatory_notification	Reporting	Regulatory deadline, incident closure
posture_summary	Reporting	Scheduled run, management request
incident_narrative	Reporting	Incident closure, analyst request
mitigation_recommendations	Reporting	Finding accepted, analyst request
analyst_query	Reporting	Analyst chat request

Dispatch is registry-driven. A specialist becomes routable by registering its workflow type; the router itself holds no agent-specific knowledge, so adding the WP3 and WP5 agents does not require changes to routing logic. An unrecognised workflow type is an explicit failure rather than a fallback to a general-purpose handler — a request that cannot be attributed to an accountable specialist should not be answered by an agent guessing at its scope.

Chaining is supported where one result implies the next question and is expected between a triage workflow and its enrichment follow-up, between an attack-simulation run and its outcome analysis, and between any finding and compliance mapping. Each chained step is a workflow, with its own enforcement and its own audit record, so a chain of five steps produces five reconstructable units rather than one opaque sequence.

9.3 Result Aggregation and Conflict Resolution

When several specialists examine the same event, the coordinator reconciles their findings before anything reaches an analyst. Agreement is the easy case: findings are linked through their related findings, and shared evidence strengthens the assessment.

Disagreement needs a rule, and the rule this design adopts is that conflict escalates rather than resolves silently. Where two agents reach materially different conclusions about the same event, the coordinator does not average their confidence scores, prefer the more confident agent, or select the more severe reading and discard the other. It presents both, with their evidence, and raises the review tier.

The reasoning is that a disagreement between specialists is information. One specialist finding a signal unremarkable while another flags it is not noise to be smoothed away — it is precisely the situation where an analyst’s judgement is worth more than an automated tie-break. Suppressing it to produce a cleaner output would discard the most useful signal the system generated.

Where findings corroborate one another, confidence is combined with weighting by agent confidence and evidence overlap, as required by R-MON-32 in D2.1; the escalation rule applies to material contradiction, not to ordinary variation between corroborating assessments.

Two ordering rules apply where findings are complementary rather than contradictory. The highest severity present sets the severity of the aggregate, so a serious finding is not diluted by routine ones alongside it. And the review tier of the aggregate is the highest tier any contributing finding carried, so combination never lowers the level of oversight applied.

9.4 HITL Interaction Model

Human oversight in this design is proportionate rather than uniform. Requiring approval for everything produces a queue nobody reads, and approval that is not read is worse than no approval, because it creates a record suggesting review that did not happen.

Table 17: Human-in-the-loop tiers

Tier	Action class	Required artefact
Tier 1	Reversible enrichment, triage, summarisation, indicator lookup, scoped retrieval	None; proceeds autonomously
Tier 2	Confidence-gated actions, medium impact, reversible containment, drafted regulatory notification	Analyst countersignature
Tier 3	High severity, low reversibility, customer-affecting, regulatory-binding, or containment with side effects	Explicit named approval with recorded justification

Four inputs determine the tier: the class of action proposed, the severity of the finding, the agent’s confidence, and the reversibility of the action. Reversibility dominates because it bounds the cost of being wrong. An enrichment lookup based on a mistaken assessment wastes analyst attention; a payment block based on the same mistake affects a customer and may itself be a reportable event.

The tier is stored on the finding and the routing rationale is written to the audit log, so a reviewer can establish not only that a decision was reviewed but why it required the review it received. Tier 2 and Tier 3 workflows pause before the action rather than after it, with state checkpointed as described in Section 5.1, and resume on approval.

Rejection is a first-class outcome. An analyst may reject a recommendation or correct it, and both are recorded against the finding: rejections feed the evaluation criteria in Section 12, and corrections feed the scoped memory in Section 5.4. This closes the loop between the two, so human judgement improves later analyses rather than being spent once and discarded.

This interaction model is shared with the analyst-facing interface specified in D2.2, which builds on the pattern developed in the email threat detection work of T4.3: an explanation traced back to the agent findings behind a verdict, followed by a prioritised set of recommended actions, with acceptance, rejection or correction recorded against the finding rather than spent once.

This tiering is the platform’s principal answer to the human-oversight expectations of the EU AI Act, discussed further in Section 11.2.

10.STRUCTURED FINDINGS AND REPORTING

Everything the orchestration layer produces leaves through one of two doors: a structured finding, or a report composed from findings. This section specifies both, together with the interfaces through which other tasks integrate.

10.1 Finding Schema Specification

A finding is the platform’s attributed analysis record. It carries not only what an agent concluded, but what it examined, which tools and model produced it, how confident it is, which obligations it touches, and what review it requires. Specialists emit findings; the coordinator, the Reporting agent, the audit log, the API and the operator console all consume them.

Making this a single validated contract rather than free-form agent output has a specific consequence: a reviewer reading a finding six months later has the same material the analyst had at the time, without needing the workflow to be re-run or the reasoning to be reconstructed from logs.

Table 18: Finding schema field groups

Group	Fields	Purpose
Identity	Finding, workflow and agent identifiers; skill packs active; timestamp	Locates the finding and what produced it
Authority	Security Context reference; entry-point origin	Establishes under whose authority the analysis ran
Assessment	Finding type; severity; confidence; status; summary; detailed analysis	The conclusion and how strongly it is held
Evidence	Evidence references; retrieval references; tool calls; model references	What the conclusion rests on and how it was obtained
Action	Recommended actions with reversibility class; review tier; related findings	What follows, and what oversight it needs
Regulatory	Mappings to DORA, NIS2, GDPR, anti-money-laundering provisions, the EU AI Act, ISO 27001, NIST CSF and PCI DSS	Which obligations the finding engages
Extension	Metadata	Forward-compatible fields

Three rules apply to every finding.

Evidence is referenced, not copied. Each reference names a locatable source such as an event in a monitored system, a file and line range, a retrieved document and section, a node in the knowledge graph. Retrieval references carry the element identifier and source reference defined in D2.4. Copying regulated content into findings would spread sensitive data into a new store with its own retention and access problems, while giving a reviewer a snapshot instead of the source.

Claims must re-ground. Assertions in the summary and analysis must trace to the evidence and retrieval references the finding carries. A finding whose narrative outruns its references is the failure mode this rule exists to catch, and the design sets a measurable target for it in Section 12, where most findings should carry non-empty evidence, retrieval references, tool calls, model references and confidence.

Validation is a single gate. Findings are persisted through one service that checks schema completeness, re-resolves the Security Context, applies the classification ceiling, writes the record and audits it. No agent writes a finding by another path. A finding that fails validation is rejected and the rejection is audited, so a malformed output is a visible event rather than a silent gap.

Findings have a lifecycle rather than being immutable at emission: open, acknowledged, accepted, rejected, corrected. Status changes are how analyst judgement attaches to the record, and rejections and corrections feed evaluation and memory respectively.

10.2 Report Types and Composition Logic

Reports serve readers who will not read findings: a supervisor receiving an incident notification, an executive reading a posture summary, an auditor reconstructing a decision. The Reporting agent composes them.

Five report types are anticipated: regulatory notifications drafted to a required format and timeline; incident narratives reconstructing what happened and what was done; posture summaries covering a period or a scope; mitigation recommendation sets; and ad-hoc analyst queries answered across the finding corpus.

Composition proceeds in three steps. The requested report type determines which findings are in scope. The scope is checked against the requesting Security Context, since report generation is an enforcement point (Section 6.1), because a report is the artefact through which material from one scope could otherwise reach a reader in another. And the selected findings are rendered into the template for that report type.

A report asserts only what its findings support. The Reporting agent composes, orders and summarises and it does not introduce assessments of its own. If a report contains a conclusion, that conclusion traces to a finding, which traces to evidence. Without this rule the reporting layer would become an unreviewed analysis path, the one place where an assertion could enter a regulatory submission without ever having passed finding validation, evidence grounding, or a review tier.

Report generation is itself audited, including the scope check, so it is possible to establish afterwards which findings a given submission drew on.

11. NON-FUNCTIONAL REQUIREMENTS AND REGULATORY ALIGNMENT

This section states the performance and deployment constraints the orchestration layer must meet, traces them to the requirements baseline in D2.1, and maps design decisions to the regulatory obligations they answer. D2.1 owns the requirements and regulatory analysis; this section does not restate either, but records what they imply for the design specified here.

11.1 Performance Requirements for the Orchestration Layer

Every target below is taken from D2.1's non-functional requirements rather than proposed here.

Table 19: Performance and deployment requirements bearing on the orchestration layer

Requirement	Target from D2.1	Design response
R-MON-07	Initial triage assessment within 60 seconds of event receipt for CRITICAL events; complete multi-agent synthesised finding within 5 minutes	Direct specialist invocation (Section 3.3), parallel dispatch (Section 9.1), bounded refinement loops (Section 9.1), per-purpose model selection (Section 7.2)
R-MON-09	Threat identification in near-real-time at point of ingestion; latency targets defined per severity level	Alerting bridge triggers workflows on arrival (Section 3.2); severity determines dispatch path
R-MON-28	Pipeline engineered for scale under peak transaction volumes, with monitored service objectives	Partitioned retrieval (Section 5.5); stateless agents permitting horizontal scaling
R-BLP-09	Concurrent request handling from both orchestrating agents to all domain agents, with stateless agent design isolating concurrent sessions	Per-request Security Context carried in workflow state (Section 5.1); no shared mutable agent state (Section 4.2)

Requirement	Target from D2.1	Design response
R-MON-02, R-MON-20	No performance degradation on production financial workloads; measured latency and CPU budget	Orchestration observes rather than intercepts; monitoring overhead sits with WP4 components
R-BLP-14	Deployable entirely on premises with no data leaving the institution's network; local inference supported	Local deployment class in the Model Gateway (Section 7.2); embedded-first storage (Section 7.3)

Three consequences follow for the design.

The 60-second budget is why the direct-invocation path exists. A critical alert whose classification is unambiguous should not queue behind coordination that adds no information, and Section 3.3 specifies that path precisely so this target is reachable without abandoning the auditability of centralised orchestration for everything else.

The five-minute budget covers automated synthesis only. Where a finding routes to Tier 2 or Tier 3, the human response time is outside the measured window. The platform's obligation is to have the synthesised finding and its evidence ready for the reviewer within five minutes, not to have obtained the reviewer's decision. This distinction matters for how the requirement is verified.

Stateless agents are a performance requirement before they are an architectural preference. R-BLP-09 requires concurrent operation across domain agents, which is achievable only if no agent holds session state between requests; this is why the Security Context and workflow state travel with each request (Section 5.1) rather than being held by the agent.

D2.1 also fixes several architectural requirements that this deliverable specifies directly.

Table 20: Architecture requirements from D2.1 specified by this design

Requirement	Subject	Specified in
R-BLP-07	Dual-mode agent interface: orchestrated and independently invokable, with enforcement and audit on every invocation	Section 4.2, Section 3.3
R-BLP-08	Per-agent vector store partitions plus a shared partition, all retrieval scoped by Security Context	Section 5.5
R-BLP-10	Governed tool adapter layer with Security Context checks, argument schema validation and append-only audit	Section 4.3
R-BLP-16	Data leakage prevention check on every agent output pipeline	Section 6.2
R-PRO-11	Prompt injection detection, input validation and sanitisation before requests reach domain agents	Section 6.2
R-REP-06	Aggregation, corroboration and conflict resolution across multi-agent outputs	Section 9.3
R-REP-10	Hallucination mitigation through evidence grounding and source attribution before delivery	Section 6.2, Section 10.1
R-REP-11	Explainable output including evidence used, confidence, source citations and auditability	Section 10.1
R-REP-13	Structured machine-readable audit trail covering routing decisions, findings with evidence, stage timings and verdicts, tamper-evident	Section 4.5
R-MON-29	Scheduled and periodic orchestration patterns alongside event-driven workflows	Section 9.1, Section 9.2
R-REP-22, R-REP-23	Event triage and classification; multi-agent parallel dispatch with partial result collection	Section 9.1, Section 9.2

11.2 Regulatory Alignment of Design Choices

D2.1 analyses the regulatory framework; what follows maps design decisions to the obligations they serve. This is an engineering account of design intent, not a legal assessment — compliance determinations rest with the deploying institutions and their advisers.

Table 21: Design responses to regulatory obligations

Instrument	Obligation engaged	Design response
DORA	ICT incident documentation, classification and reporting to competent authorities within defined timelines	Append-only audit trail with routing decisions and stage timings (Section 4.5); regulatory mappings on findings (Section 10.1); regulatory notification reports (Section 10.2)
NIS2	Essential-entity incident reporting; supply chain risk management	Alerting bridge capturing events across monitoring tooling (Section 3.2); code and pipeline analysis (Section 8.2)
GDPR	Data minimisation; purpose limitation; erasure; accountability	Evidence held by reference rather than copy (Section 10.1); scoped memory with bounded retention and audited purge (Section 5.4); namespace- and scope-partitioned retrieval (Section 5.5)
EU AI Act	Human oversight proportionate to risk; transparency of AI-supported decisions	Tiered review model with recorded rationale (Section 9.4); model references and evidence on every finding (Section 10.1)

Three design properties carry most of this weight, and they are worth naming because they are what a supervisory reviewer will test.

Attribution. Every finding names the evidence it rests on, the tools that produced it and the model that generated it. This single property serves DORA’s documentation obligations, the EU AI Act’s transparency expectations, and R-REP-11’s explainability requirement, because all three ultimately ask the same question: on what basis was this decision made?

Proportionate oversight. The tiered model in Section 9.4 ties the depth of human review to the reversibility and impact of the action, which is the structure the EU AI Act’s human-oversight provisions call for. Uniform review would satisfy the letter and defeat the purpose.

Bounded data movement. Referencing rather than copying evidence, partitioning knowledge by namespace and data scope, bounding memory retention, and supporting fully local inference together mean regulated data stays where its controls already apply. This is what makes the GDPR position defensible and what makes on-premises deployment (R-BLP-14) achievable rather than aspirational.

One limitation should be stated. These properties make compliance demonstrable; they do not make it automatic. A deployment that configures overly broad scopes, disables review tiers, or routes inference to an unapproved provider can be built on this design and still fail its obligations. The design constrains what agents can do; it does not constrain what an operator can configure.

12.VALIDATION APPROACH

This section defines beforehand the conditions to measure the correctness of the proposed LLM Orchestration architecture in CyberAId once implemented. The criteria below are written so that an implementation either meets them or does not.

Table 22: Evaluation criteria and pass thresholds

Dimension	Question	Metric	Pass criterion
Architecture readiness	Can a new organisation be onboarded without code changes?	Lines of code changed; manual configuration touchpoints	Zero lines changed; all changes confined to runtime configuration and the organisation profile
Architecture readiness	Can the same workflow run from every entry point?	Workflow completion rate across entry points, per workflow	At least 95% completion on canonical workflows, with no entry-point-specific code paths
Architecture readiness	Can the model provider be swapped without agent changes?	Code difference under provider swap, one local and one private API	Difference confined to the Model Gateway; agent definitions unchanged
Architecture readiness	Can extension packs be enabled independently?	Service restarts required to enable a pack	No more than one restart; install and enable recorded as audit events
Security and governance	Are tool calls deterministic and audited?	Schema-validation rejection rate; audit trace coverage	100% rejection of out-of-schema arguments; every executed call audited
Security and governance	Is review-tier routing correct?	Macro-F1 against labelled action-class to tier mappings	At least 0.9 macro-F1 on the scenario set
Output quality	Are findings fully attributed?	Proportion of findings carrying non-empty evidence, retrieval references, tool calls, model references and confidence	At least 95%
Output quality	Are reports free of unsupported claims?	Proportion of report claims traceable to a finding or evidence	At least 90% supported
Output quality	Do analyst corrections reach scoped memory without contaminating evidence?	Memory-write isolation tests	100% of writes confined to scoped memory namespaces
Operational	How long does diagnosing a failed workflow take from the audit trail alone?	Wall-clock time to root cause on benchmark failures	Under 5 minutes
Operational	Is connector health reporting complete?	Proportion of connectors emitting health events on the expected cadence	100%
Operational	Is scheduled-workflow latency stable?	Launch latency under nominal load	95th percentile under 15 minutes; 50th percentile under 1 minute

13.CONCLUSIONS AND NEXT STEPS

D2.5 specifies the design of the CyberAid LLM orchestration layer presenting how agents are built, what authorises them, how they hold state, what they produce, and who reviews their output before it becomes an action.

The central decision is the one that shapes everything else. Rather than treating each specialist agent as an independent system with its own security posture, this design puts a single canonical architecture underneath all of them, with enforcement, evidence handling, observability and audit provided by the runtime rather than implemented per agent. The consequence is practical: the agents specified in Section 8 will be built by tasks spread across three work packages, and under this design that distribution adds domain capability without multiplying the number of places a regulator, an auditor or a security reviewer has to examine.

Six things are now settled for the technical work that builds against this deliverable. The Security Context and its enforcement points define what any agent may reach. The finding schema defines what every agent produces and what a claim must be grounded in. The tiered review model defines when a human decides rather than an agent. The state and context policy defines what persists, what is compacted, and what may never be summarised away. The entry-point model and the alerting bridge that carries events from security tools into the LLM layer define how work arrives. And the validation criteria in Section 12 define, in advance, what a correct implementation looks like.

Section 8 states domains and boundaries rather than each specialist's tool inventory, reasoning strategy or realisation pattern. Those choices are made by the teams building each agent, against the shared contract this deliverable defines.

The immediate next step is the realisation of the design itself: the shared agent runtime and its Runtime Core, including Security Context enforcement, the Model Gateway, the Tool Adapter Layer and the Context Broker over the Knowledge and Data Fabric, together with the coordinator, the finding service, the audit log and the tiered review mechanism. Initial agents built on that runtime will exercise the contract end to end, and the agent catalogue, workflow registration and finding schema will then allow the specialist agents to be developed in parallel and to cooperate through the coordinator rather than through bespoke integrations. Alongside this, the open design decisions recorded in this deliverable will be closed as implementation proceeds: the three storage choices in Table 8, the remaining agent realisation patterns in Section 8.4, and whether forensic analysis is exposed through a dedicated agent to close the incident-response and forensics gap identified in Section 2.4. The validation criteria in Section 12 will then be applied to the implemented platform.

Finally, the proposed design makes the platform's behaviour inspectable, so that what an agent concluded, on whose authority it acted, on what evidence it relied and who reviewed it all remain visible after the fact. This does not make an agent's judgement correct, nor does it prevent a deployment from being configured in ways that undermine its own guarantees. What it does ensure is that when an agent is wrong, and in this domain some proportion of the time it will be, the error is visible, attributable and correctable, rather than an unexplained output that an analyst must either trust or discard. For financial institutions operating under DORA, NIS2 and the EU AI Act, that property is the precondition for using this technology at all.

REFERENCES

- [1] G. Fatouros, K. Mavrogiorgos, G. Makridis, J. Soldatos and D. Kyriazis, "LLM-Based Agents for Cybersecurity: A Systematic Review of Architectures, Applications, and Open Challenges," *Journal of Cybersecurity and Privacy*, vol. 6, no. 5, p. 156, August 2026.
- [2] G. Fatouros, G. Makridis, G. Kousiouris, J. Soldatos and D. Kyriazis, "An Organization-Scoped LLM Agent Runtime Architecture for Regulated Cybersecurity Operations," in *3rd Workshop on Next Generation IoT and AI Systems for Trusted, Human-Centered Intelligence, co-located with DCOSS-IoT 2026*, Reykjavik, Iceland, 2026.
- [3] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," in *International Conference on Learning Representations (ICLR)*, Kigali, Rwanda, 2023.
- [4] Anthropic, "Model Context Protocol Specification," 2024. [Online]. Available: <https://modelcontextprotocol.io>.
- [5] P. Lewis, E. Perez and A. Piktus, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [6] European Parliament and Council, "Regulation (EU) 2022/2554 on Digital Operational Resilience for the Financial Sector (DORA)," Official Journal of the European Union, Brussels, Belgium, 2022.
- [7] P. Drammeh, "Multi-Agent LLM Orchestration Achieves Deterministic, High-Quality Decision Support for Incident Response," *arXiv preprint arXiv:2511.15755*, 2025.
- [8] Y. Zhu, A. Kellermann, A. Gupta, P. Li, R. Fang, R. Bindu and D. Kang, "Teams of LLM Agents Can Exploit Zero-Day Vulnerabilities," in *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics*, 2026.
- [9] C. Song, L. Ma, J. Zheng, J. Liao, H. Kuang and L. Yang, "Audit-LLM: Multi-Agent Collaboration for Log-Based Insider Threat Detection," *arXiv preprint arXiv:2408.08902*, 2024.
- [10] T. Wu, S. Yang, S. Liu, D. Nguyen, S. Jang and A. Abuadbba, "ThreatModeling-LLM: Automating Threat Modeling Using Large Language Models for Banking System," *arXiv preprint arXiv:2411.17058*, 2024.
- [11] European Parliament and Council, "Directive (EU) 2022/2555 on Measures for a High Common Level of Cybersecurity across the Union (NIS 2 Directive)," Official Journal of the European Union, 2022.
- [12] European Parliament and Council, "Regulation (EU) 2024/1689 Laying Down Harmonised Rules on Artificial Intelligence (AI Act)," Official Journal of the European Union, Brussels, Belgium, 2024.
- [13] G. Fatouros, G. Makridis, J. Soldatos and D. Kyriazis, "CyberAId: AI-Driven Cybersecurity for Financial Service Providers," *arXiv preprint arXiv:2605.01892*, 2026.