

# NSC Caelus AI - System Specification (v1)

Generated 2026-02-01

This document specifies the NSC Caelus AI architecture as an audit-grade, tool-using, tag-routed system with gates/rails and tiered memory. It is written to be testable: every major component has measurable acceptance criteria and required artifacts (receipts).

## 1. Mission Lock

Objective: Deliver a reliable 'smart solver' that can (a) plan and execute tool-using workflows, (b) preserve causality and safety constraints via gates/rails, and (c) accumulate audit-grade evidence for every meaningful claim.

Primary KPI	Tool-use success rate (offline + shadow)
Target	$\geq 0.85$ over 7-day window
Guardrail 1	Hallucination rate $\leq 0.03$
Guardrail 2	p95 latency $\leq 1.8$ seconds
Constraints	Deterministic replay; strict schema validation; privacy scope enforcement
Operating env	Offline evaluation -> shadow mode -> online A/B with rollback

## 2. System Overview

Caelus is organized as a layered control system: orchestration (PhaseLoom), enforcement (Gates/Rails), memory (Aeonic tiers), and evidence capture (Receipt Ledger). NSC tags provide a shared 'semantic bus' between layers, enabling fast routing, slice-aware evaluation, and policy enforcement.

### 2.1 Core components

PhaseLoom Orchestrator: schedules work across concurrent threads (planning, retrieval, tool calls, verification) and enforces deterministic replay via fixed decoding parameters, stable tool contracts, and run receipts.

Gates/Rails: guards that either allow, warn, block, reroute, or defer actions. Rails are hard invariants (e.g., no-future-leak) that cannot be overridden by the model.

Aeonic Memory Bank: a tiered memory system with explicit retention, privacy policy, and provenance for each record.

Receipt Ledger: immutable log of inputs, outputs, hashes, environment, and metrics, used to support claims in the experiment DAG.

### 3. Definitions Ledger

The definition ledger is the single source of truth for: task definitions, ground truth semantics, dataset versions/splits, model interfaces, and evaluation protocol. Changes to the ledger require a receipt and version bump.

#### 3.1 Canonical task types

Planning: produce an executable plan with explicit tool steps and stop conditions.

Retrieval: fetch external or internal artifacts; must include provenance references.

Generation: synthesize text/code/artifacts; must declare uncertainty if evidence is missing.

Verification: validate outputs via tests, hash checks, invariants, or independent solvers.

#### 3.2 Model interface contract

Inputs: (a) user request, (b) current context + allowed tools, (c) NSC tag context, (d) relevant memory slices (explicit).

Outputs: (a) user-facing result, (b) internal tool calls, (c) receipts and tag events, (d) updated memory records (if permitted).

### 4. NSC Tag System

NSC tags are short, composable labels that annotate: requests, intermediate states, tool calls, decisions, artifacts, and claims. Tags are stable identifiers (ASCII) in a small ontology. They are not free-form 'vibes'; each tag has a meaning and allowed attachment targets.

#### 4.1 Tag algebra

Tags support intersection and implication rules. Example: `safety:causality` implies `gate_no_future_leak` must run on every retrieval and tool output. `gate:verified` implies schema and hash checks passed.

Routing rule example: if `domain=math_physics` and tags include `nsc:claim`, prioritize verification thread and require numeric checks before final response.

#### 4.2 Tag lifecycle

Creation: tags originate from deterministic rules (preferred) or human annotation. Every tag creation emits a `TagEvent` with evidence and confidence.

Decay: tags expire by TTL policy unless promoted by receipts to retained class.

Audit: all tag events are queryable and replayable from receipts.

## 5. Gates and Rails

Gates produce structured decisions (GateDecision) that can alter or stop execution. Rails are non-negotiable safety and correctness invariants implemented as always-on gates.

### 5.1 Required gates

gate\_schema\_validation: every tool output and memory write must validate against schema.

gate\_no\_future\_leak: prohibits using information timestamped after the task context time.

gate\_privacy\_scope: enforces PII classification and allowed\_scopes; blocks unsafe writes or disclosures.

gate\_dimensional\_consistency (physics domains): checks unit consistency and range sanity for numeric outputs.

### 5.2 Gate outcomes

allow: proceed. warn: proceed but annotate and require verification step. block: stop and return safe failure. reroute: switch tool or strategy. defer: request human review or more information.

## 6. Aeonic Memory Bank

Memory is explicit, tiered, and policy-governed. The system never 'implicitly remembers' something without creating a MemoryRecord that has provenance and retention.

### 6.1 Tiers

L0 Ephemeral: per-step scratch; discarded after completion.

L1 Working: session memory; compaction allowed.

L2 Project: long-lived context; requires receipts for promotion; used for ongoing engineering work.

L3 Canonical: audited facts and definitions; write access is gated and rare.

### 6.2 Write policy

All writes require: schema validation, provenance refs, privacy classification, and retention policy. Writes without these are blocked.

### 6.3 Compaction

Compaction is a transformation with its own receipt: input record hashes -> output record hash, plus a diff summary and loss budget.

## 7. Receipt Ledger and Experiment DAG

Every meaningful result must be backed by a receipt. Receipts are immutable records that bind: datasets, code, configs, seeds, and produced artifacts.

## 7.1 Node types

Dataset node: source -> filtering -> dedupe -> split -> hash.

Model node: architecture -> init -> training recipe -> checkpoint id.

Eval node: benchmark -> prompts -> scorer -> aggregated metrics.

Claim node: statement + KPI + guardrails + conditions + evidence refs.

## 7.2 Acceptance criteria

No claim may be marked 'tested' without: (a) an EvalRun artifact, (b) a Receipt referencing that run, (c) schema-valid hashes for all inputs/outputs, and (d) guardrails within limits.

## 8. Reference Integration: Smart GR Solver

The smart GR solver is a reference domain integration demonstrating: deterministic scheduling, constraint enforcement, and audit-grade outputs. It is treated as a high-rigor domain where invariants are explicit (constraints, gauge conditions, stability limits).

### 8.1 Required properties

Deterministic replay: same initial conditions + same code/config seed -> statistically consistent results within tolerance.

Constraint checks: Hamiltonian and momentum constraint norms logged each step; gate triggers on divergence.

Evidence capture: every simulation run emits receipts with parameter hashes, code commit, and resulting metrics.

## 9. Deployment Reality

Caelus is deployed as a service with strict observability. Production mode requires shadow testing, alerting, and rollback paths.

### 9.1 Monitoring requirements

See runbooks/deployment\_observability.md for minimum dashboards and alerts.

### 9.2 Rollback plan

Rollback is triggered when primary KPI drops below threshold or a P0 safety gate violation occurs. Rollback must restore the last known-good model\_id and routing policy bundle, and freeze L2/L3 memory writes until audit completes.

## 10. References

Glenn, Jerome. Foresight on Demand: 'Foresight Towards the 2nd Strategic Plan for Horizon Europe' - Artificial General Intelligence: Issues and Opportunities (Rapid Exploration). The Millennium Project, Feb 2023.

Wang, Pei. The Evaluation of AGI Systems. Temple University preprint (undated).